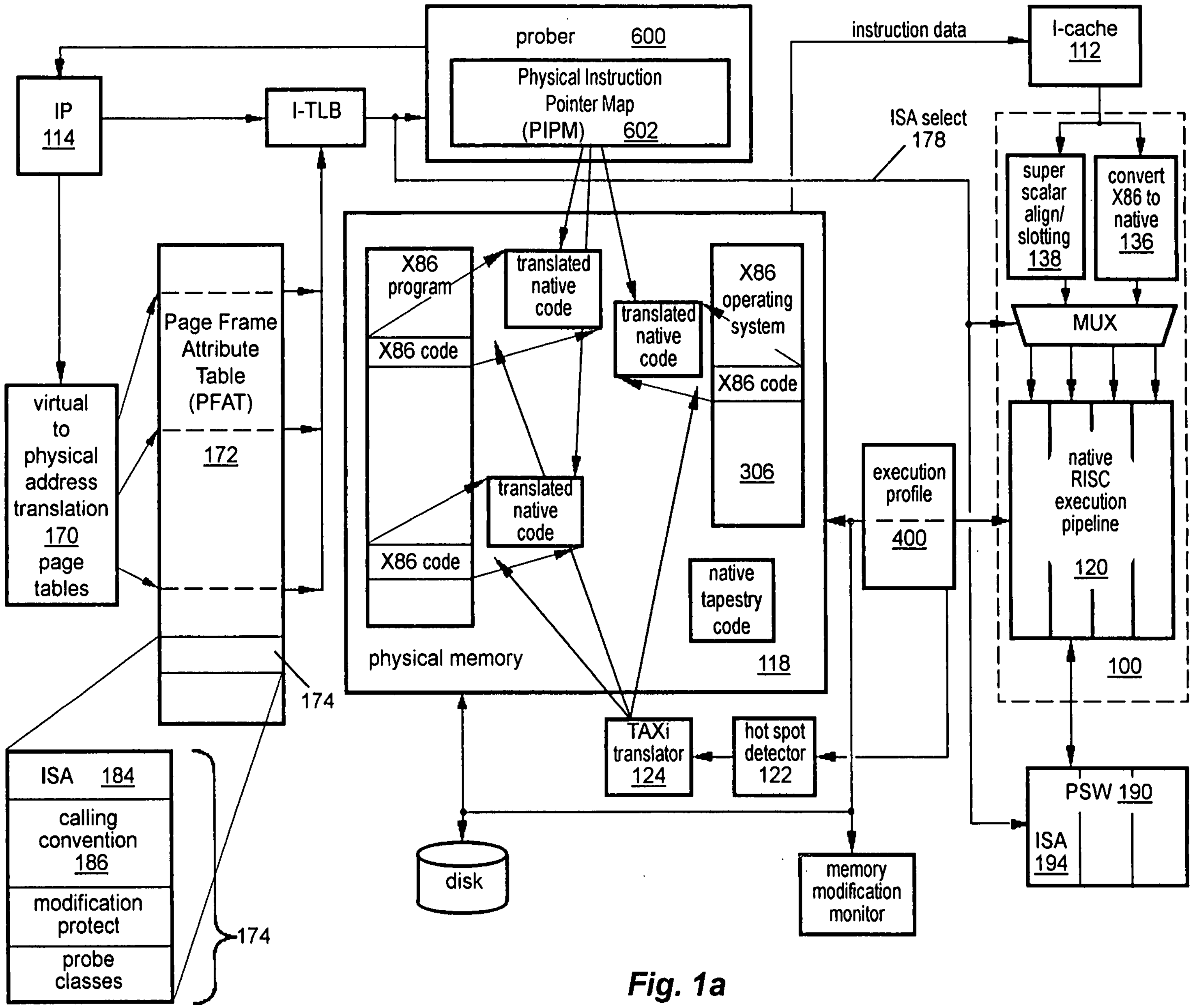


Fig. 1a

09/434,374

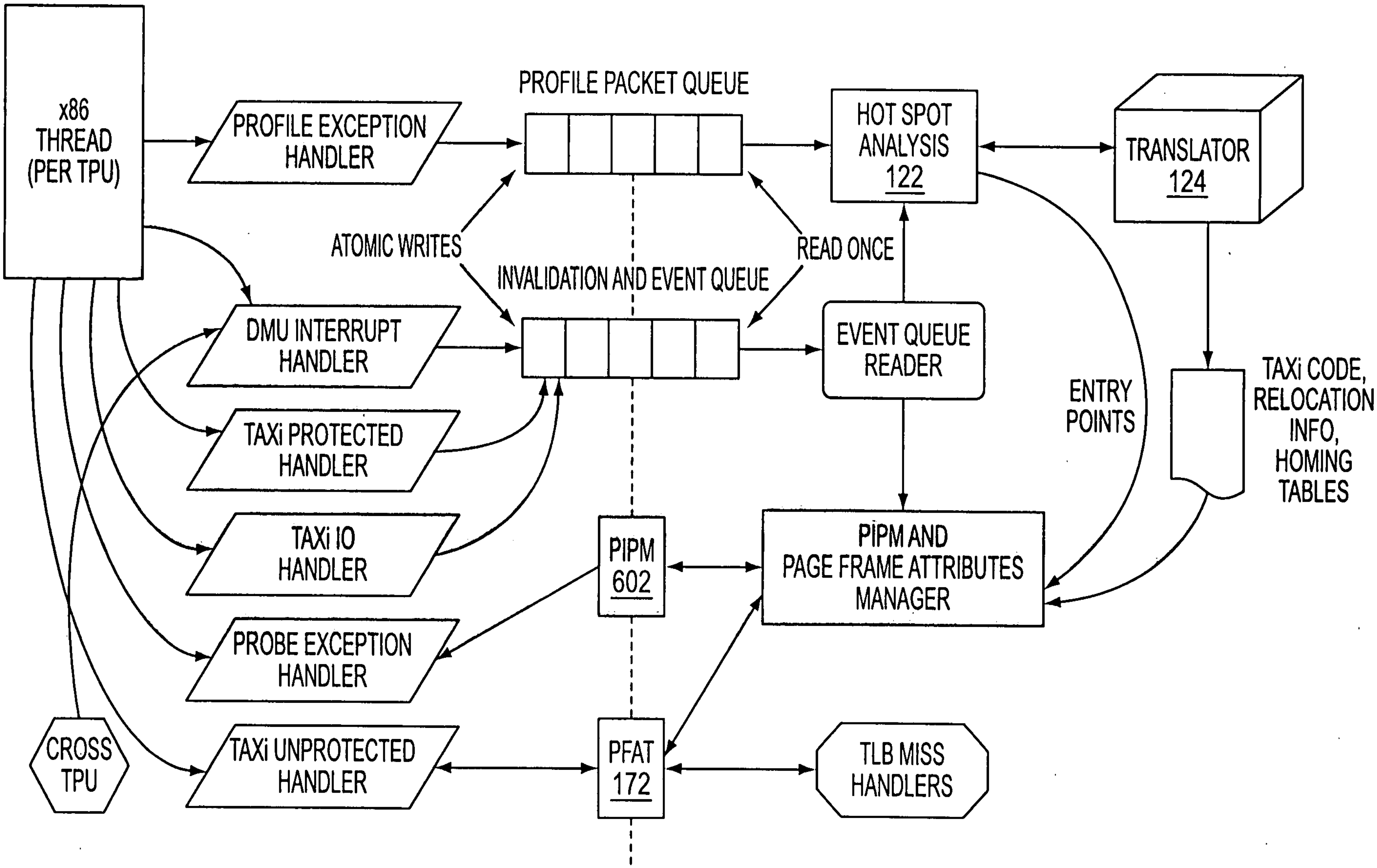
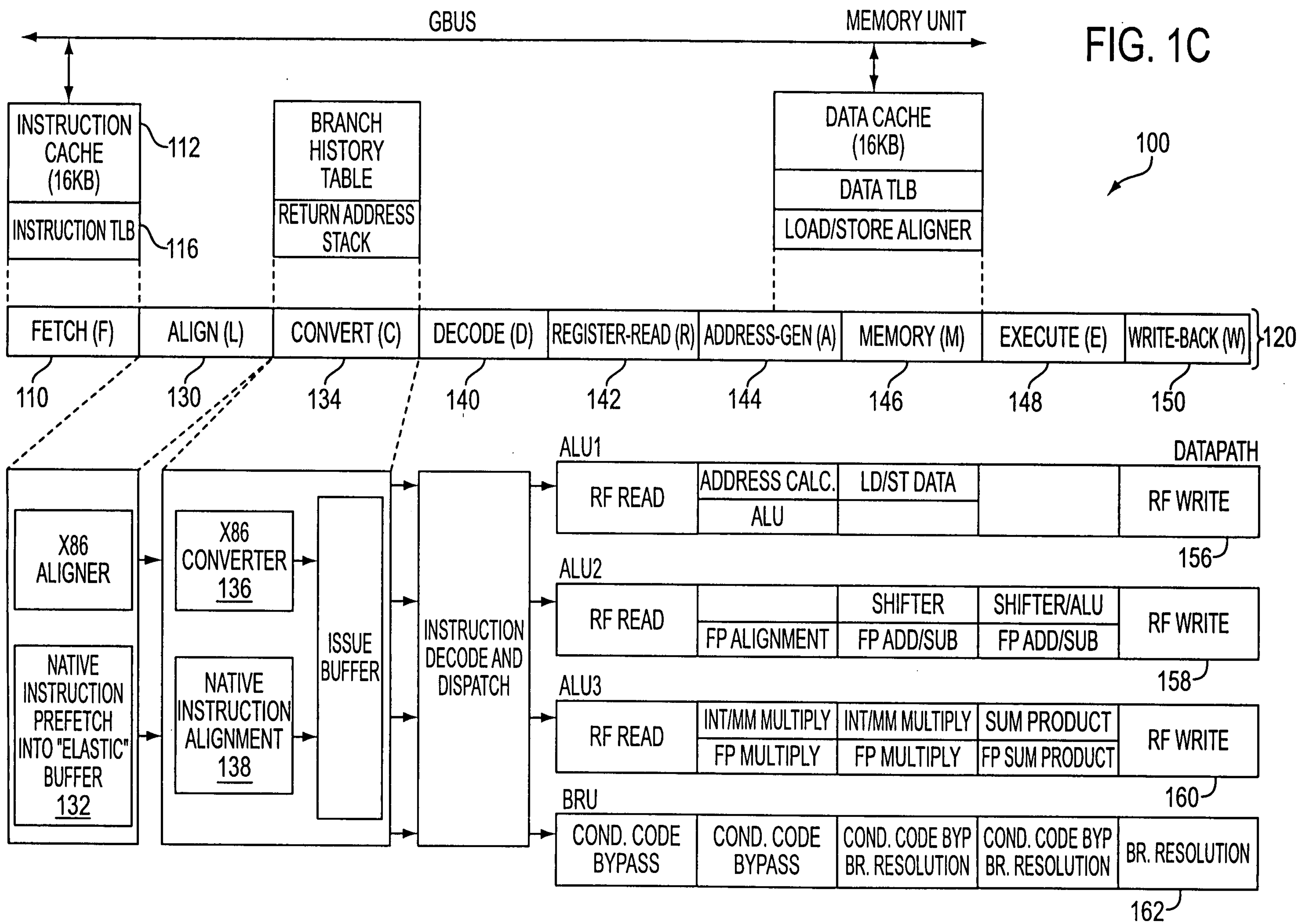

x86
THREAD
(PER TPU)
PROFILE EXCEPTION
HANDLER
DMU INTERRUPT
HANDLER
TAXi PROTECTED
HANDLER
TAXi IO
HANDLER
PROBE EXCEPTION
HANDLER
TAXi UNPROTECTED
HANDLER
CROSS
TPU
ATOMIC WRITES
PROFILE PACKET QUEUE
INVALIDATION AND EVENT QUEUE
READ ONCE
PIPM
602
PFAT
172
HOT SPOT
ANALYSIS
122
EVENT QUEUE
READER
PIPM AND
PAGE FRAME ATTRIBUTES
MANAGER
TLB MISS
HANDLERS
TRANSLATOR
124
ENTRY
POINTS
TAXi CODE,
RELOCATION
INFO,
HOMING
TABLES

FIG. 1B

FIG. 1C



GBUS
MEMORY UNIT
INSTRUCTION CACHE (16KB)
112
INSTRUCTION TLB
116
BRANCH HISTORY TABLE
RETURN ADDRESS STACK
DATA CACHE (16KB)
DATA TLB
LOAD/STORE ALIGNER
100
FETCH (F)
ALIGN (L)
CONVERT (C)
DECODE (D)
REGISTER-READ (R)
ADDRESS-GEN (A)
MEMORY (M)
EXECUTE (E)
WRITE-BACK (W)
120
110
130
134
140
142
144
146
148
150
X86 ALIGNER
NATIVE INSTRUCTION PREFETCH INTO "ELASTIC" BUFFER 132
X86 CONVERTER 136
NATIVE INSTRUCTION ALIGNMENT 138
ISSUE BUFFER
INSTRUCTION DECODE AND DISPATCH
ALU1
DATAPATH
RF READ
ADDRESS CALC.
ALU
LD/ST DATA
RF WRITE
156
ALU2
RF READ
FP ALIGNMENT
SHIFTER
FP ADD/SUB
SHIFTER/ALU
FP ADD/SUB
RF WRITE
158
ALU3
RF READ
INT/MM MULTIPLY
FP MULTIPLY
INT/MM MULTIPLY
FP MULTIPLY
SUM PRODUCT
FP SUM PRODUCT
RF WRITE
160
BRU
COND. CODE BYPASS
COND. CODE BYPASS
COND. CODE BYP BR. RESOLUTION
COND. CODE BYP BR. RESOLUTION
BR. RESOLUTION
162

</ 434,517

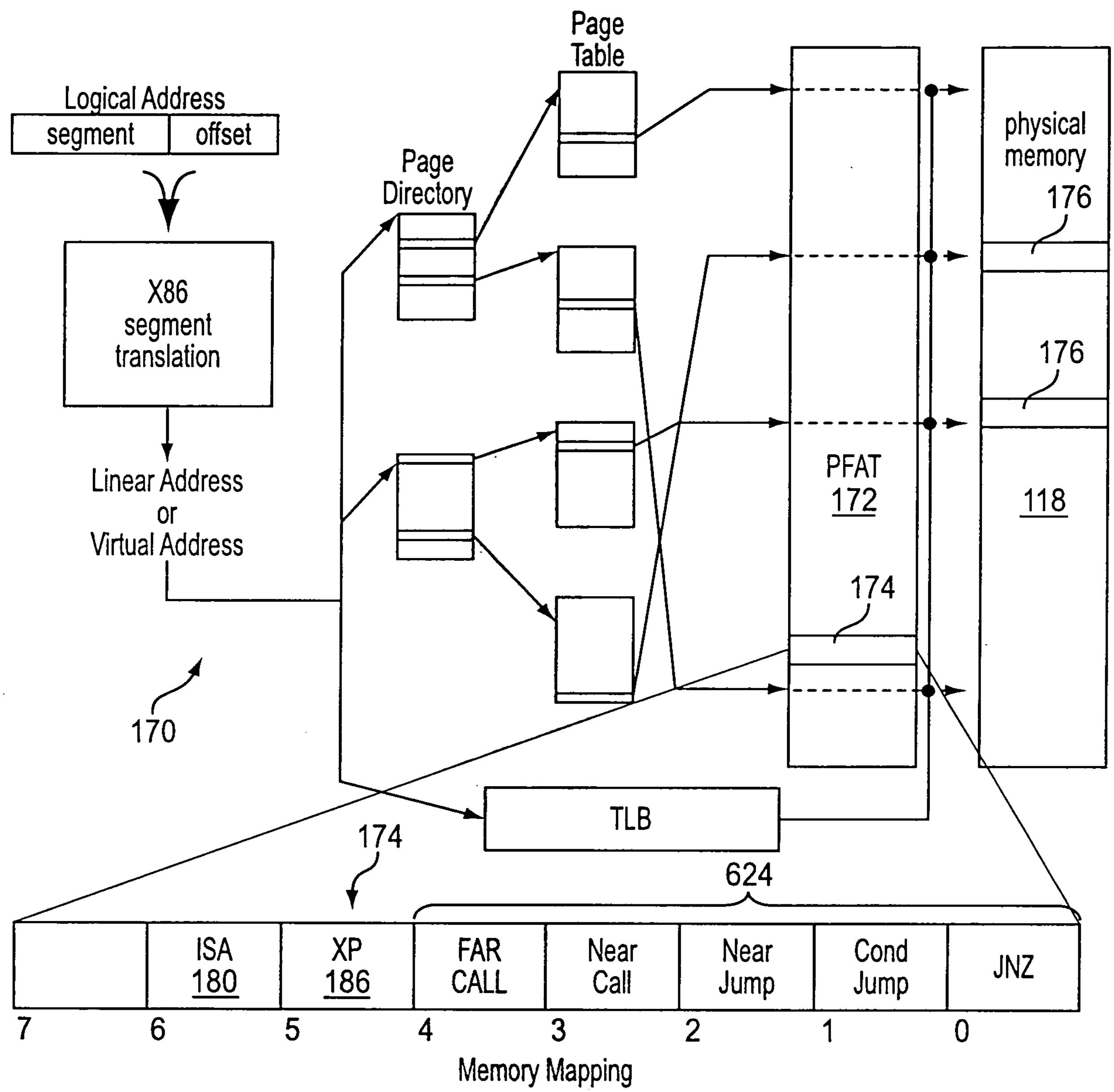


FIG. 1D

09/434,374

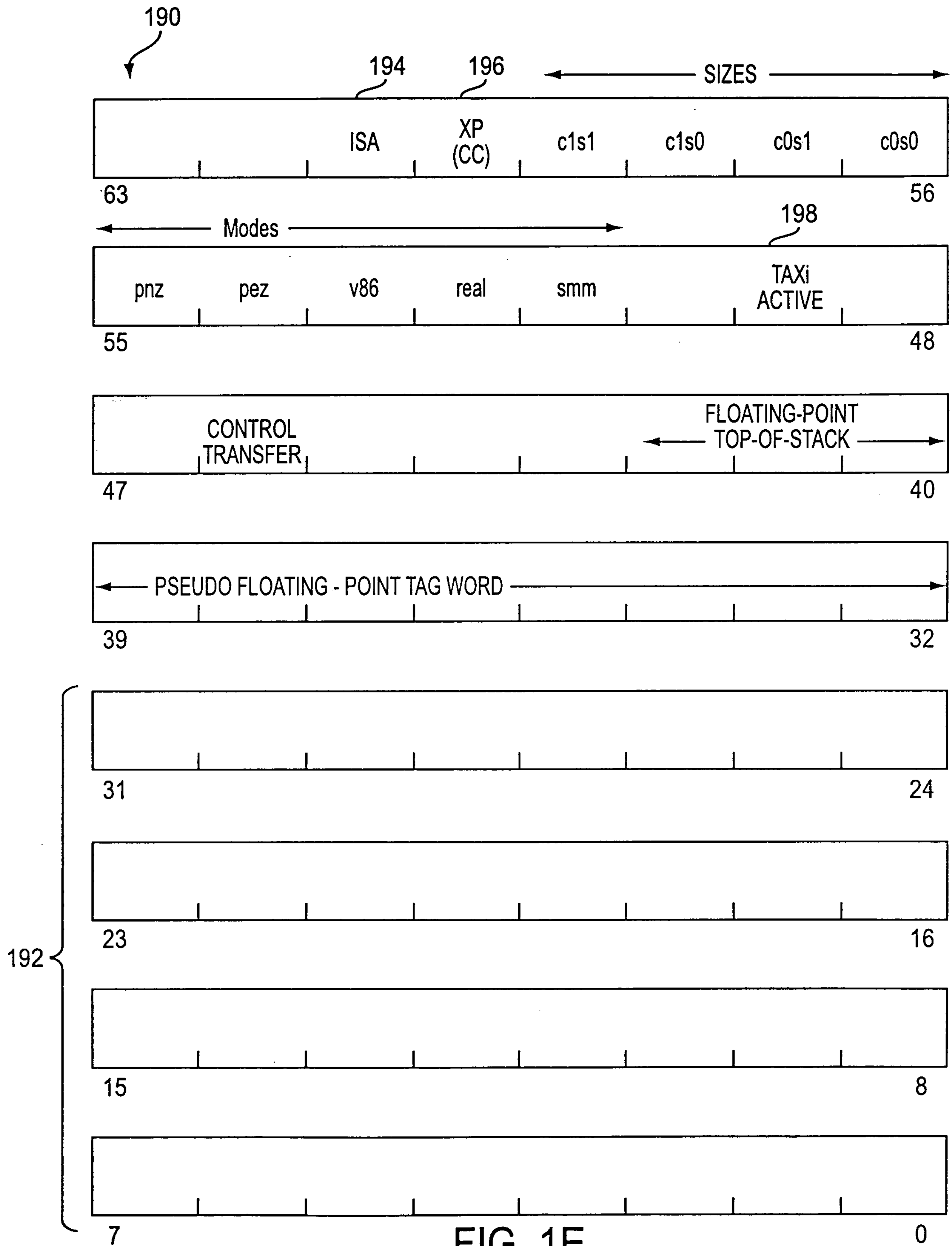


FIG. 1E

0714 34, 344

I-TLB PROPERTY BITS	DECODED PROPERTY VALUES		PROTECTED	INTERPRETATION	INSTRUCTIONS SENT TO:	COLLECT PROFILE TRACE-PACKETS?	PROBE FOR TRANSLATED CODE	I/O MEMORY REFERENCE EXCEPTIONS
	ISA 194	CC 200						
00	TAP	TAP	NO	NATIVE CODE OBSERVING NATIVE RISCy CALLING CONVENTIONS	NATIVE DECODER	NO	NO	FAULT IF SEG.tio
01	TAP	x86	NO	NATIVE CODE OBSERVING x86 CALLING CONVENTIONS	NATIVE DECODER	NO	NO	FAULT IF SEG.tio
10	x86	x86	NO	x86 CODE, UNPROTECTED - *TAXI* PROFILE COLLECTION ONLY	x86 HW CONVERTER	IF ENABLED	NO	TRAP IF PROFILING
11	x86	x86	YES	x86 CODE, PROTECTED - *TAXI* CODE MAY BE AVAILABLE	x86 HW CONVERTER	IF ENABLED	BASED ON I-TLB PROBE ATTRIBUTES	TRAP IF PROFILING

180,182, 184,186

184,186

FIG. 2A

204

	TRANSITION (SOURCE => DEST) ISA & CC PROPERTY VALUES	HANDLER ACTION
212	00 => 00	NO TRANSITION EXCEPTION
214	00 => 01	VECT_xxx_X86_CC EXCEPTION - HANDLER CONVERTS FROM NATIVE TO x86 CONVENTIONS
216	00 => 1x	VECT_xxx_X86_CC EXCEPTION - HANDLER CONVERTS FROM NATIVE x86 CONVENTIONS, SETS UP EXPECTED EMULATOR AND PROFILING STATE
218	01 => 00	VECT_xxx_TAP_CC EXCEPTION - HANDLER CONVERTS FROM x86 TO NATIVE CONVENTIONS
220	01 => 01	NO TRANSITION EXCEPTION
222	01 => 1x	VECT_X86_ISA EXCEPTION [CONDITIONAL BASED ON PCW.X86_ISA_ENABLE FLAG] - SETS UP EXPECTED EMULATOR AND PROFILING STATE
224	1x => 00	VECT_xxx_TAP_CC EXCEPTION - HANDLER CONVERTS FROM x86 TO NATIVE CONVENTIONS
226	1x => 01	VECT_TAP_ISA EXCEPTION [CONDITIONAL BASED PCW.TAP_ISA_ENABLE FLAG] - NO CONVENTION CONVERSION NECESSARY
228	1x => 10	NO TRANSITION EXCEPTION - [PROFILE COMPLETE POSSIBLE, PROBE POSSIBLE]
230	1x => 11	NO TRANSITION EXCEPTION - [PROFILE COMPLETE POSSIBLE, PROBE NOT POSSIBLE]

FIG. 2B

	NAME	DESCRIPTION	TYPE
242	VECT_call_X86_CC	PUSH ARGS, RETURN ADDRESS, SET UP x86 STATE	FAULT ON TARGET INSTRUCTION
244	VECT_jump_X86_CC	SET UP x86 STATE	FAULT ON TARGET INSTRUCTION
246	VECT_ret_no_fp_X86_CC	RETURN VALUE TO EAX:EDX, SET UP x86 STATE	FAULT ON TARGET INSTRUCTION
248	VECT_ret_fp_X86_CC	RETURN VALUE TO x86 FP STACK, SET UP x86 STATE	FAULT ON TARGET INSTRUCTION
250	VECT_call_TAP_CC	x86 STACK ARGS, RETURN ADDRESS TO REGISTERS	FAULT ON TARGET INSTRUCTION
252	VECT_jump_TAP_CC	x86 STACK ARGS TO REGISTERS	FAULT ON TARGET INSTRUCTION
254	VECT_ret_no_fp_TAP_CC	RETURN VALUE TO RV0	FAULT ON TARGET INSTRUCTION
256	VECT_ret_any_TAP_CC	RETURN TYPE UNKNOWN, SETUP RV0 AND RVDP	FAULT ON TARGET INSTRUCTION

FIG. 2C

09/434,577

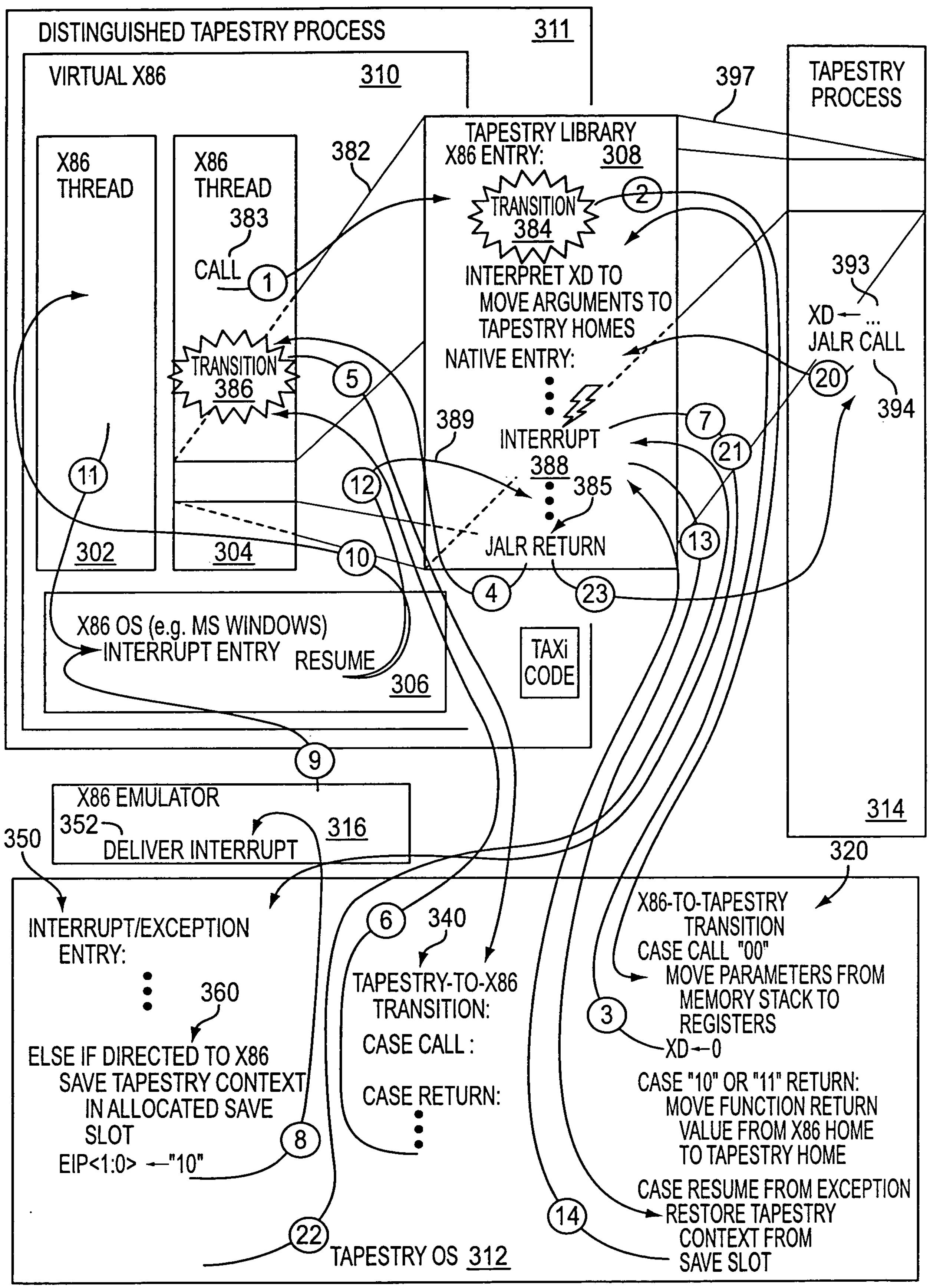


FIG. 3A

09/434,514

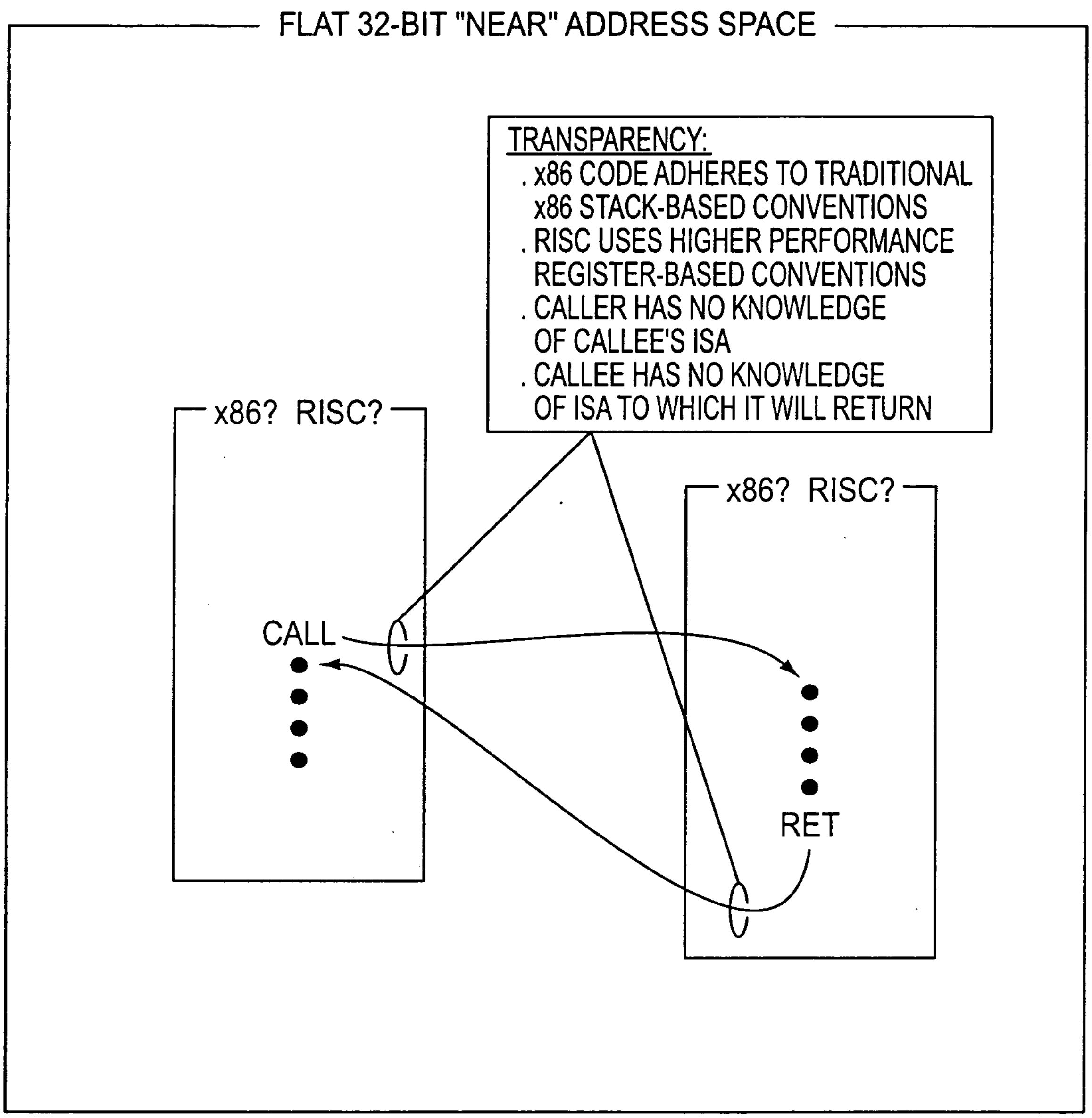


FIG. 3B

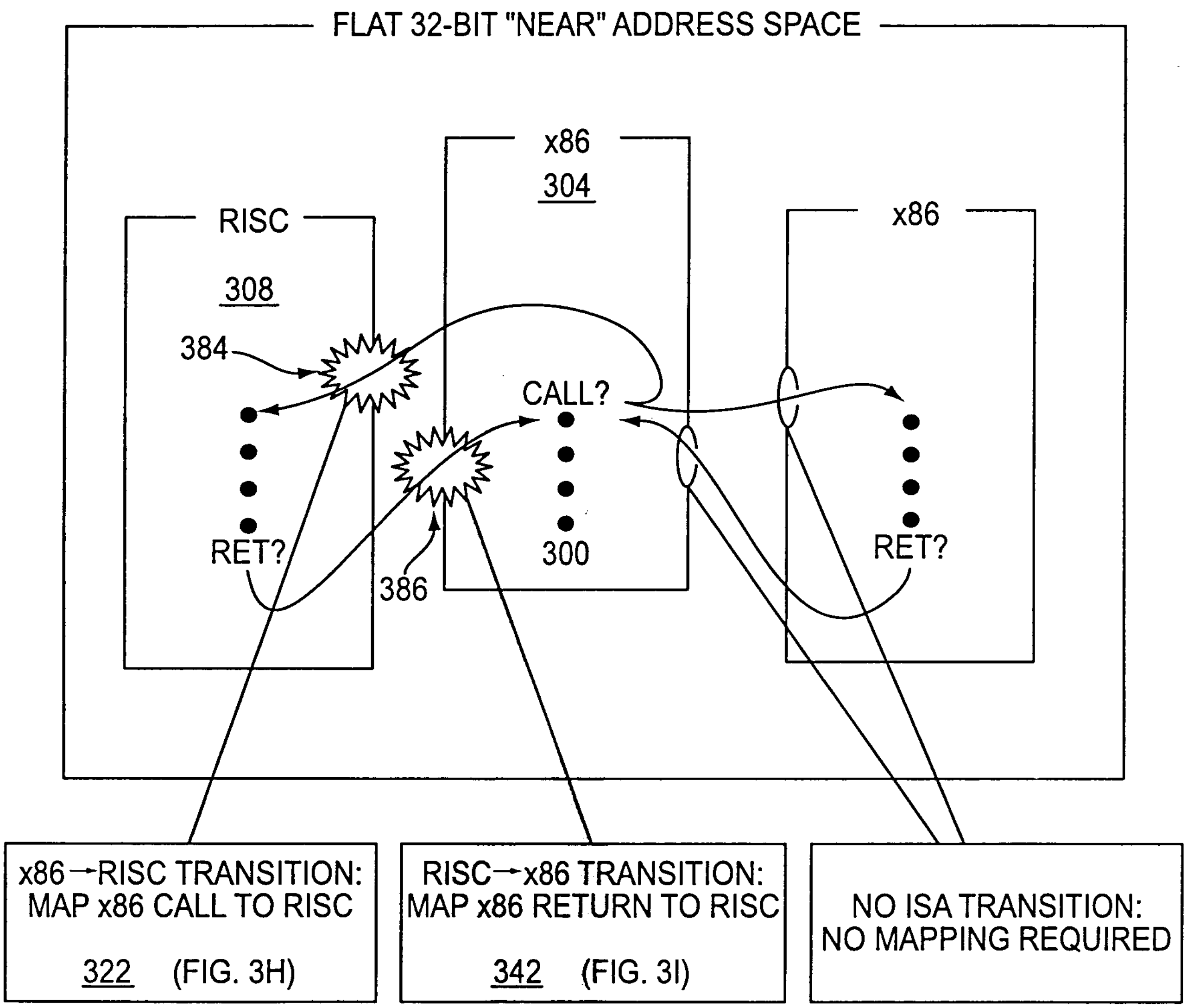

FLAT 32-BIT "NEAR" ADDRESS SPACE
x86
304
RISC
308
x86
384
CALL?
RET?
300
386
RET?
x86→RISC TRANSITION: MAP x86 CALL TO RISC
322 (FIG. 3H)
RISC→x86 TRANSITION: MAP x86 RETURN TO RISC
342 (FIG. 3I)
NO ISA TRANSITION: NO MAPPING REQUIRED

FIG. 3C

09/434,374

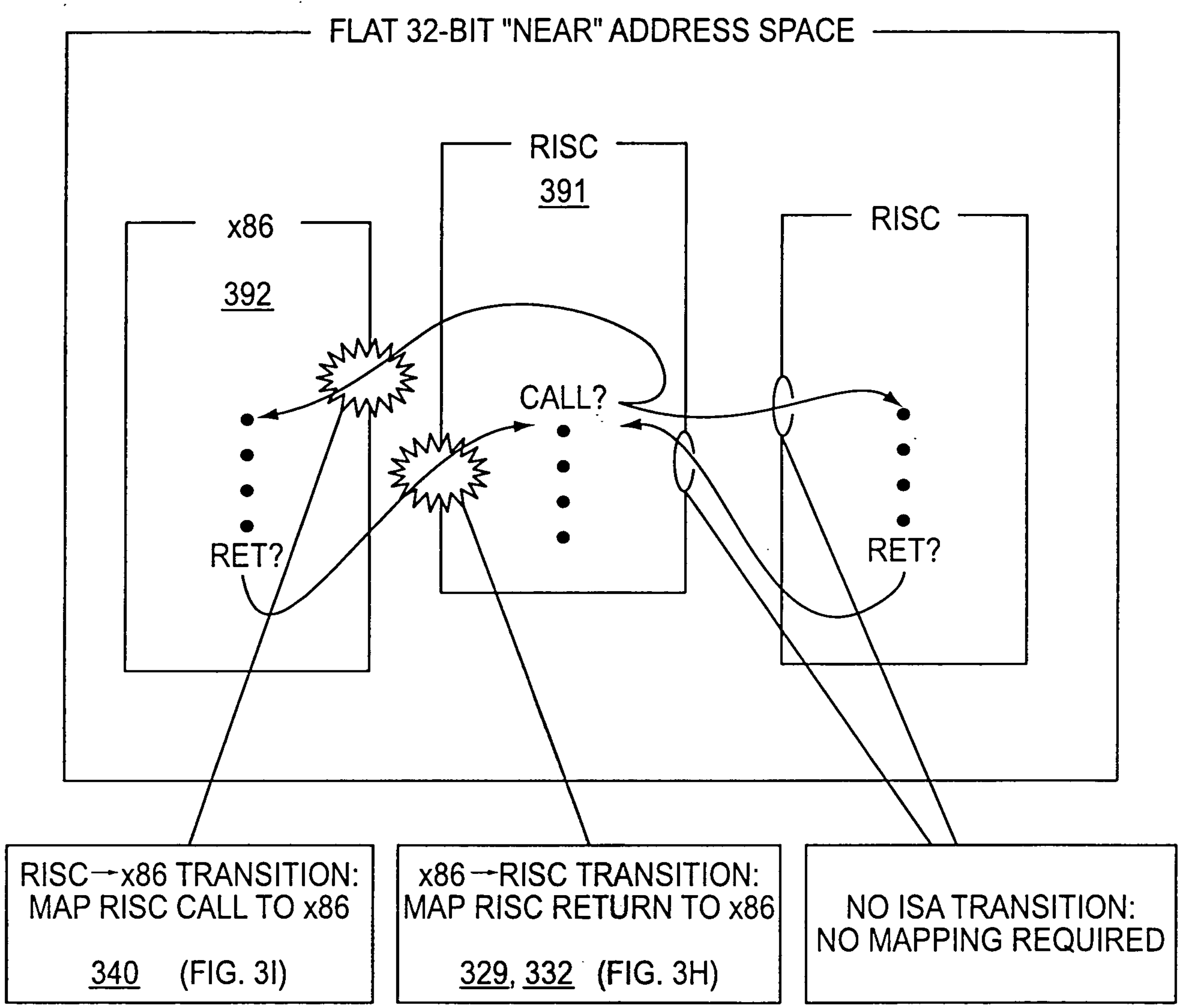


FIG. 3D

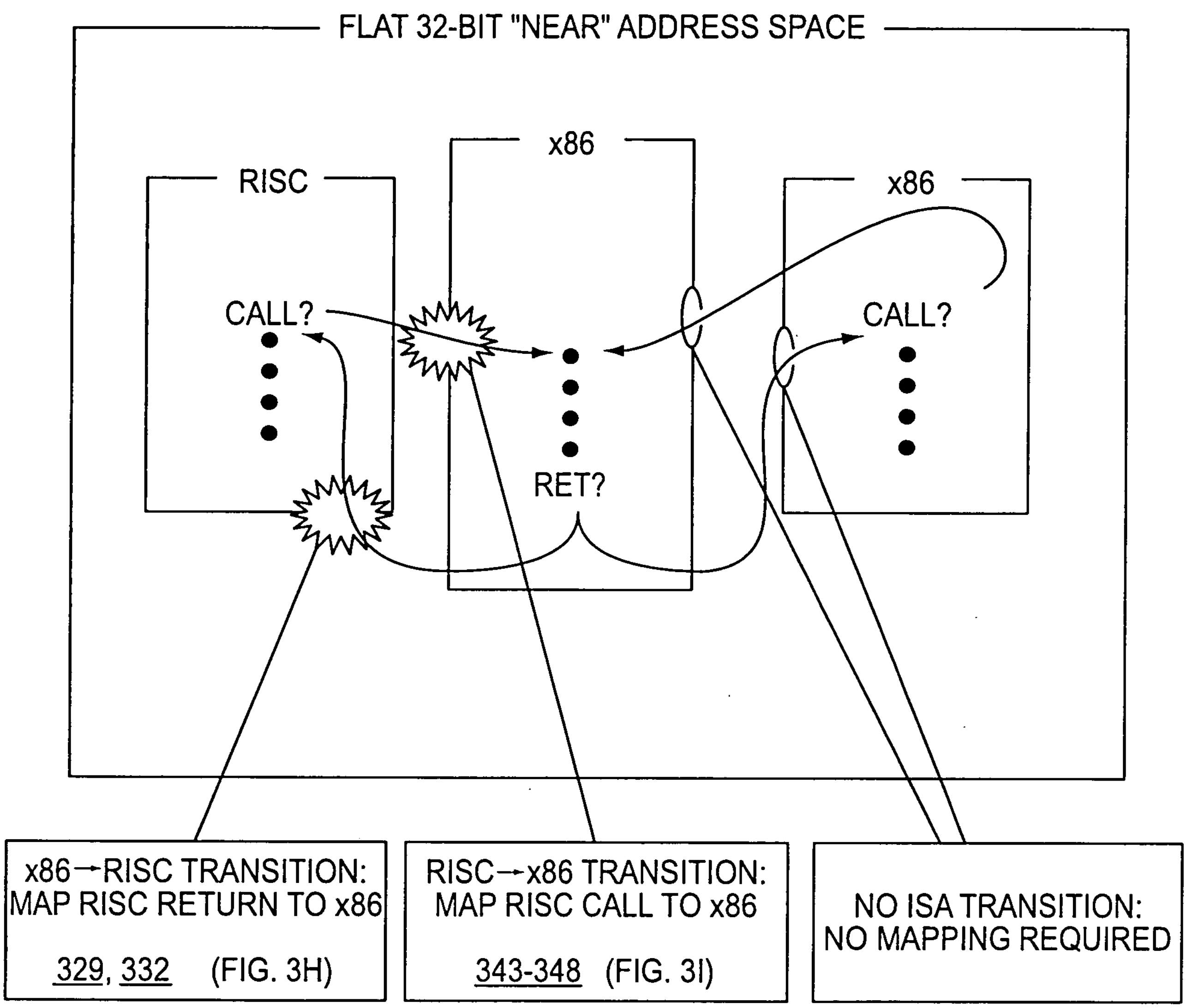
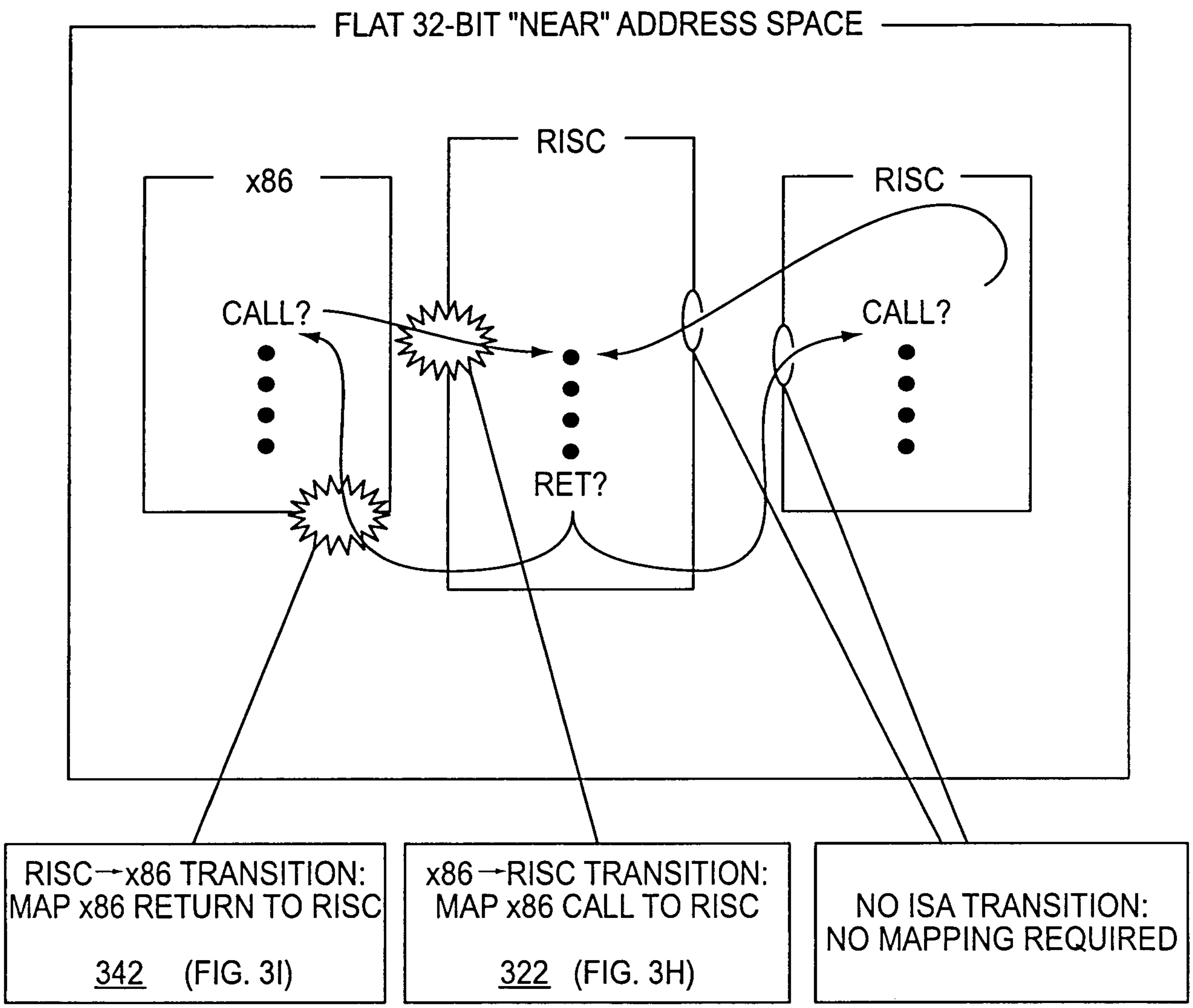

FLAT 32-BIT "NEAR" ADDRESS SPACE
RISC
x86
x86
CALL?
CALL?
RET?
x86→RISC TRANSITION:
MAP RISC RETURN TO x86
329, 332 (FIG. 3H)
RISC→x86 TRANSITION:
MAP RISC CALL TO x86
343-348 (FIG. 3I)
NO ISA TRANSITION:
NO MAPPING REQUIRED

FIG. 3E



FLAT 32-BIT "NEAR" ADDRESS SPACE
x86
RISC
RISC
CALL?
CALL?
RET?
RISC→x86 TRANSITION:
MAP x86 RETURN TO RISC
342 (FIG. 3I)
x86→RISC TRANSITION:
MAP x86 CALL TO RISC
322 (FIG. 3H)
NO ISA TRANSITION:
NO MAPPING REQUIRED

FIG. 3F

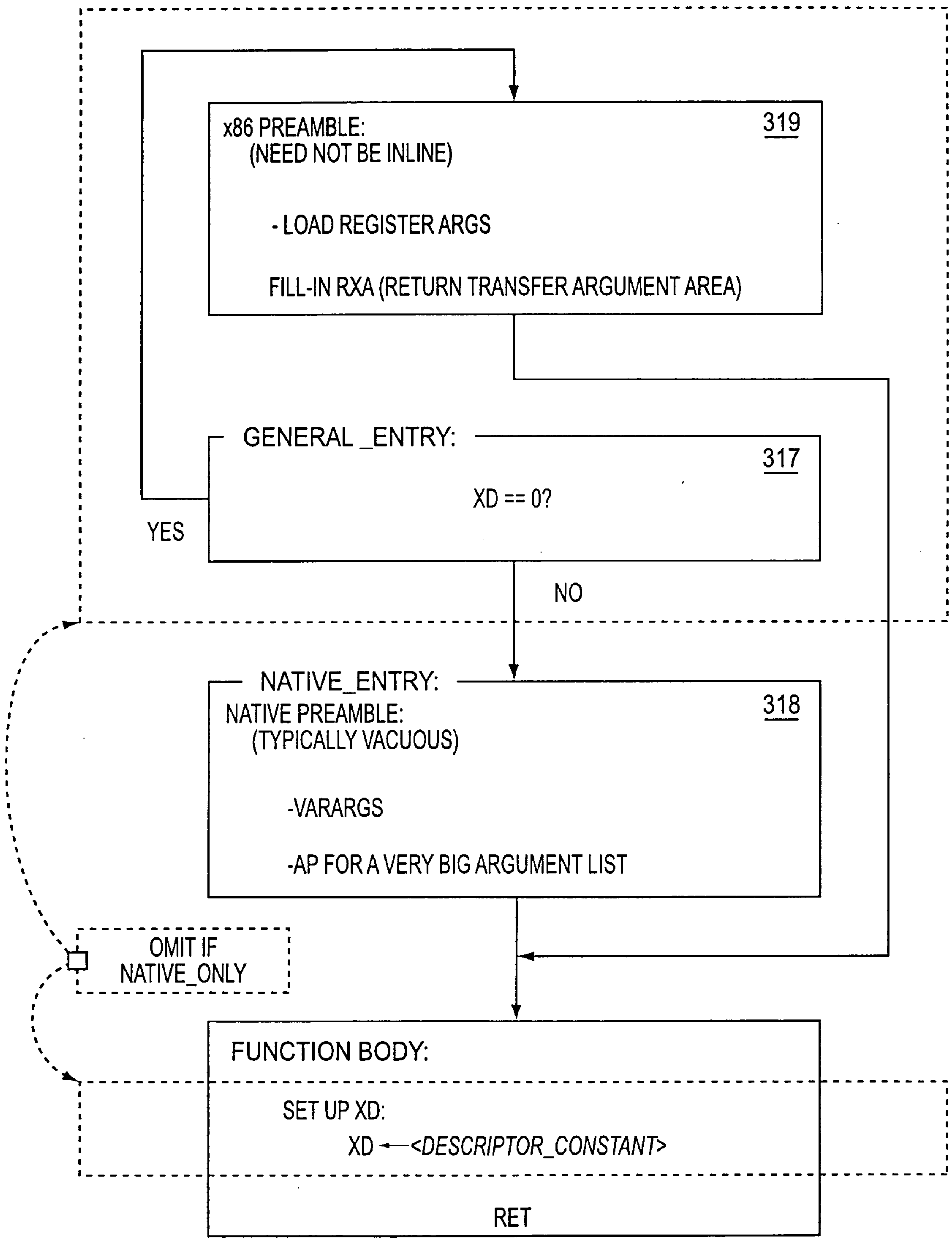

x86 PREAMBLE:
(NEED NOT BE INLINE)
319
- LOAD REGISTER ARGS
FILL-IN RXA (RETURN TRANSFER ARGUMENT AREA)
GENERAL _ENTRY:
317
XD == 0?
YES
NO
NATIVE_ENTRY:
NATIVE PREAMBLE:
(TYPICALLY VACUOUS)
318
-VARARGS
-AP FOR A VERY BIG ARGUMENT LIST
OMIT IF
NATIVE_ONLY
FUNCTION BODY:
SET UP XD:
XD ←<DESCRIPTOR_CONSTANT>
RET

FIG. 3G

09/434,394

320

X86-to Tapestry transition exception handler

```
// This handler is entered under the following conditions:
// 1. An x86 caller invokes a native function
// 2. An x86 function returns to a native caller
// 3. x86 software returns to or resumes an interrupted native function following
//    an external asynchronous interrupt, a processor exception, or a context switch
        321
dispatch on the two least-significant bits of the destination address      {
case"00"        // calling a native subprogram                                          322
    // copy linkage and stack frame information and call parameters from the memory
    // stack to the analogous Tapestry registers
    LR ← [SP++]          // set up linkage register 323
    AP ← SP              // address of first argument 324
    SP ← SP - 8          // allocate return transfer argument area 326
    SP ← SP & (-32)      // round the stack pointer down to a 0 mod 32 boundary 327
    XD ← 0               // inform callee that caller uses X86 calling conventions 328
case "01"        // resuming an X86 thread suspended during execution of a native routine   370
    if the redundant copies of the save slot number in EAX and EDX do not match or if
          the redundant copies of the timestamp in EBX:ECX and ESI:EDI do not match {  371
          // some form of bug or thread corruption has been detected
          goto TAPESTRY_CRASH_SYSTEM( thread-corruption-error-code ) 372
    }
    save the EBX:ECX timestamp in a 64-bit exception handler temporary register
           (this will not be overwritten during restoration of the full native context)  373
    use save slot number in EAX to locate actual save slot storage 374
    restore full entire native context (includes new values for all x86 registers) 375
    if save slot's timestamp does not match the saved timestamp { 376
          // save slot has been reallocated; save slot exhaustion has been detected
          goto TAPESTRY_CRASH_SYSTEM( save-slot-overwritten-error-code ) 377
    }
    free the save slot 378
case"10"        // returning from X86 callee to native caller, result already in registers   332
    RV0<63:32> ← edx<31:00>          // in case result is 64 bits 333
    convert the FP top-of-stack value from 80 bit X86 form to 64-bit form in RVDP 334
    SP ← ESI                          // restore SP from time of call 337
case"11"        // returning from X86 callee to native caller, load large result from memory   329
    RV0..RV3 ← load 32 bytes from [ESI-32] // (guaranteed naturally aligned) 330
    SP ← ESI                          // restore SP from time of call 337
}
EPC ← EPC & -4          // reset the two low-order bits to zero 336
RFE 338
```

FIG. 3H

340

```
Tapestry-to-X86 transition exception handler
    // This handler is entered under the following conditions:
    // 1. a native caller invokes an x86 function
    // 2. a native function returns to an x86 caller
    switch on XD<3:0> {   341

    XD_RET_FP:                        // result type is floating point
        FO/FI ← FINFLATE.de( RVDP)    // X86 FP results are 80 bits
        SP ← from RXA save            // discard RXA, pad, args
        FPCW ← image after FINIT & push // FP stack has 1 entry
        goto EXIT

    XD_RET_WRITEBACK:                 // store result to @RVA, leave RVA in eax
        RVA ← from RXA save           // address of result area
        copy decode(XD<8:4>) bytes from RV0..RV3 to [RVA]            342
        eax ← RVA                     // X86 expects RVA in eax
        SP ← from RXA save            // discard RXA, pad, args
        FPCW ← image after FINIT      // FP stack is empty
        goto EXIT

    XD_RET_SCALAR:                    // result in eax:eda
        edx<31:00> ← eax<63:32>       // in case result is 64 bits
        SP ← from RXA save            // discard RXA, pad, args
        FPCW ← image after FINIT      // FP stack is empty
        goto EXIT

    XD_CALL_HIDDEN_TEMP:  // allocate 32 byte aligned hidden temp   343
        esi ← SP                      // stack cut back on return
        SP ← SP - 32                  // allocate max size temp     344
        RVA ← SP                      // RVA consumed later by RR
        LR<1:0> ← "11"                // flag address for return & reload   345
        goto CALL_COMMON

    default:                          // remaining XD_CALL_xxx encodings
        esi ← SP                      // stack cut back on return   343
        LR<1:0> ← "10"                // flag address for return    346
CALL_COMMON:
        interpret XD to push and/or reposition args   347
        [--SP] ← LR                   // push LR as return address
EXIT:                                                               348
        setup emulator context and profiling ring buffer pointer
    }
    RFE   349                         // to original target
}
```

FIG. 3I

09/434,394

350

```
interrupt/exception handler of Tapestry operating system:
    // Control vectors here when a synchronous exception or asynchronous interrupt is to be
    // exported to / manifested in an x86 machine.

// The interrupt is directed to something within the virtual X86, and thus there is a possibility
// that the X86 operating system will context switch. So we need to distinguish two cases:
//   either the running process has only X86 state that is relevant to save, or
//   there is extended state that must be saved and associated with the current machine context
//        (e.g., extended state in a Tapestry library call in behalf of a process managed by X86 OS)
if execution was interrupted in the converter – EPC.ISA == X86 {
        // no dependence on extended/native state possible, hence no need to save any     351
        goto EM86_Deliver_Interrupt( interrupt-byte )
} else if EPC.Taxi_Active {
        // A Taxi translated version of some X86 code was running. Taxi will rollback to an
        // x86 instruction boundary. Then, if the rollback was induced by an asynchronous external
        // interrupt, Taxi will deliver the appropriate x86 interrupt. Else, the rollback was induced
        // by a synchronous event so Taxi will resume execution in the converter, retriggering the     353
        // exception but this time with EPC.ISA == X86
        goto TAXi_Rollback( asynchronous-flag, interrupt-byte )
} else if EPC.EM86 {
        // The emulator has been interrupted. The emulator is coded to allow for such
        // conditions and permits re-entry during long running routines (e.g. far call through a gate)     354
        // to deliver external interrupts
        goto EM86_Deliver_Interrupt( interrupt-byte )
} else {
        // This is the most difficult case - the machine was executing native Tapestry code on
        // behalf of an X86 thread. The X86 operating system may context switch. We must save
        // all native state and be able to locate it again when the x86 thread is resumed.
                361
        allocate a free save slot; if unavailable free the save slot with oldest timestamp and try again
        save the entire native state (both the X86 and the extended state)
        save the X86 EIP in the save slot     362
        overwrite the two low-order bits of EPC with "01" (will become X86 interrupt EIP)     363     360
        store the 64-bit timestamp in the save slot, in the X86 EBX:ECX register pair (and,
                for further security, store a redundant copy in the X86 ESI:EDI register pair)     364
        store the a number of the allocated save slot in the X86 EAX register (and, again for
                further security, store a redundant copy in the X86 EDX register)     365
        goto EM86_Deliver_Interrupt( interrupt-byte )
}                                              369
```

FIG. 3J

```
typedef struct {
    save_slot_t *       newer,              // pointer to next-most-recently-allocated save slot
    save_slot_t *       older;              // pointer to next-older save slot
    unsigned int64      epc;                // saved exception PC/IP
    unsigned int64      pcw;                // saved exception PCW (program control word)
    unsigned int64      registers[63];      // save the 63 writeable general registers
    ...                                     // other words of Tapestry context
    timestamp_t         timestamp;          // timestamp to detect buffer overrun
    int                 save_slot_ID;       // ID number of the save slot
    boolean             save_slot_is_full;      // full / empty flag
} save_slot_t;

save_slot_t *       save_slot_head;         // pointer to the head of the queue
save_slot_t *       save_slot_tail;         // pointer to the tail of the queue

system initialization
    reserve several pages of unpaged memory for save slots
```

FIG. 3K

09/434,394

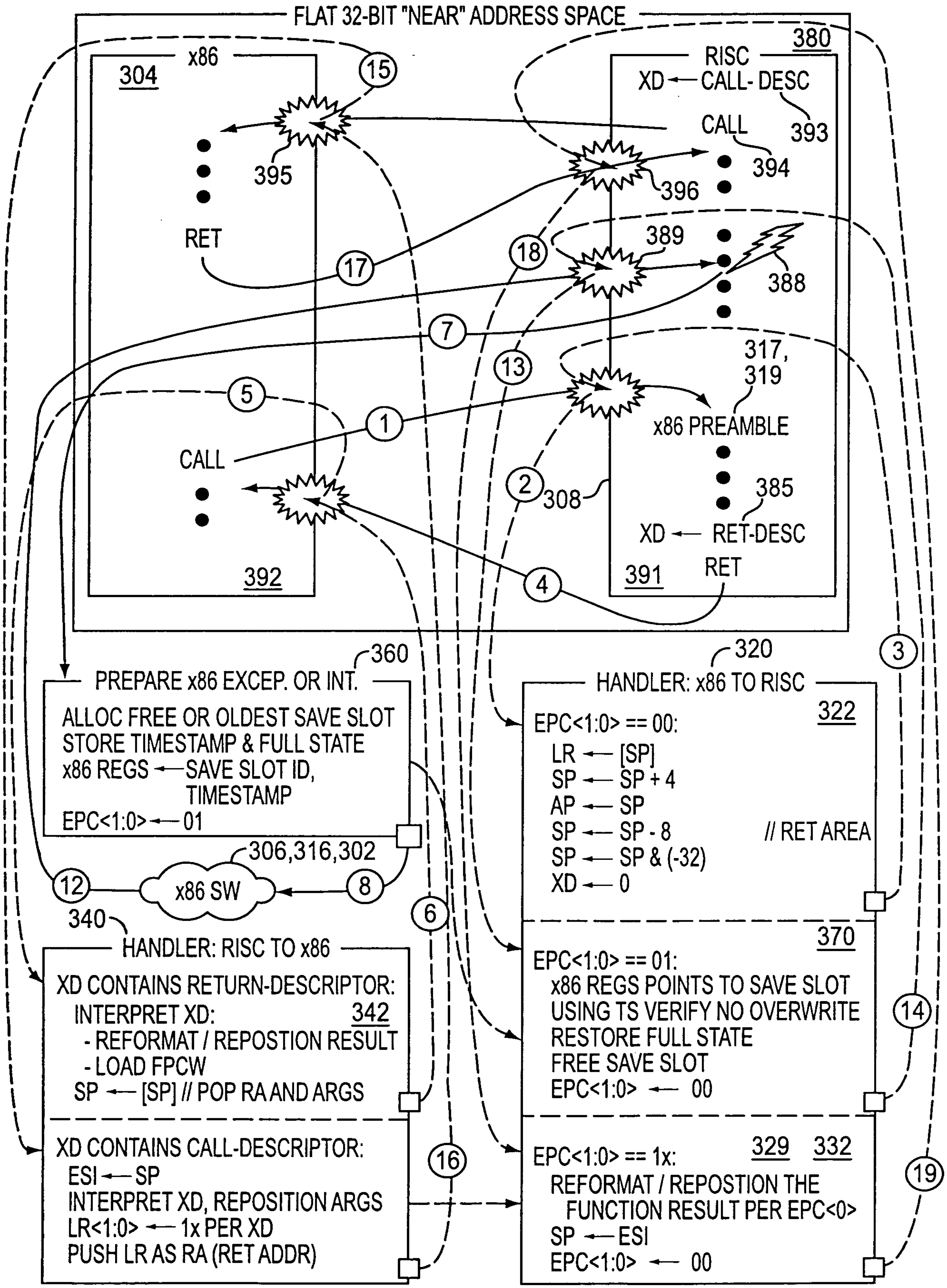


FIG. 3L

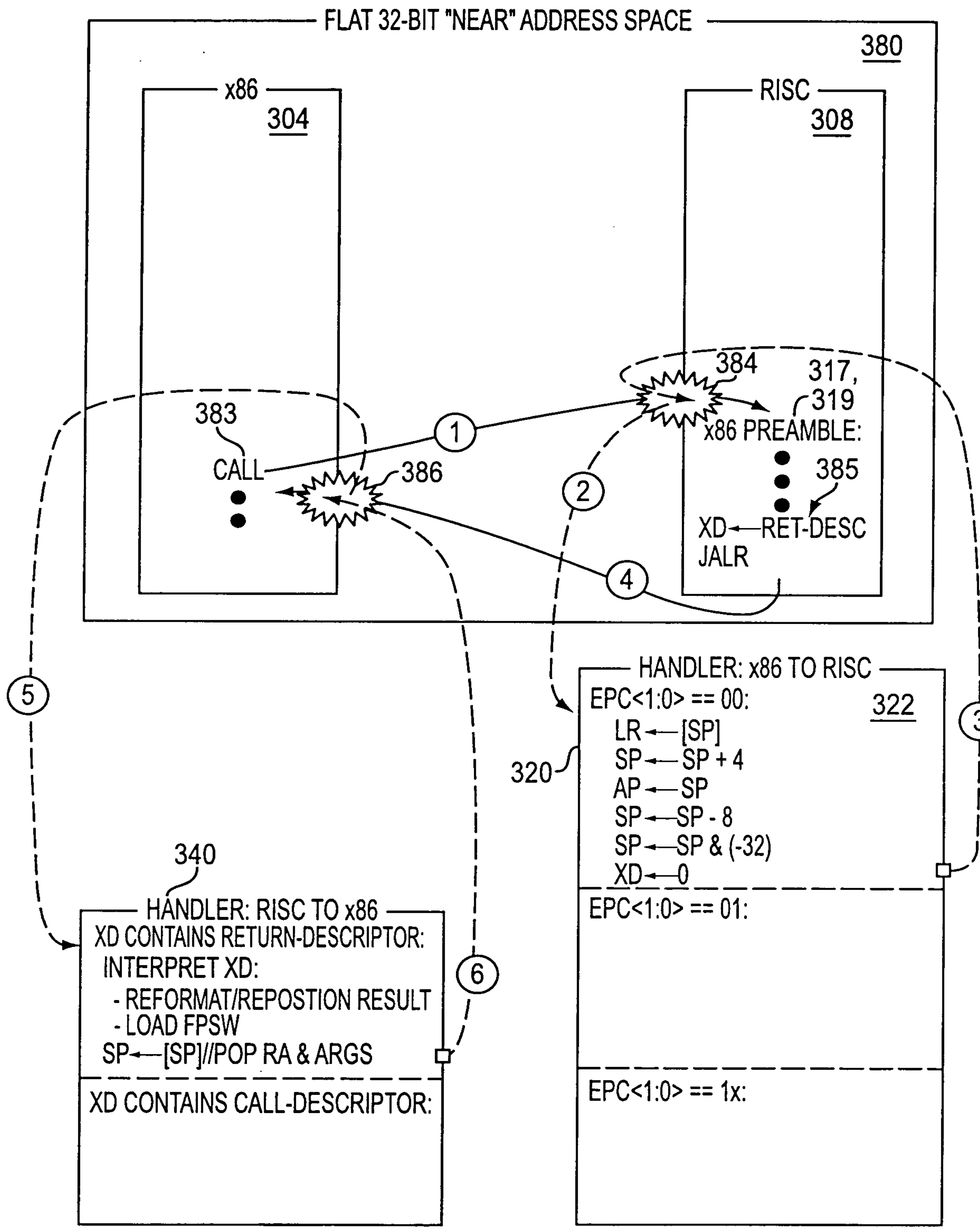

FLAT 32-BIT "NEAR" ADDRESS SPACE
380
x86
304
RISC
308
383
CALL
386
384
317,
319
x86 PREAMBLE:
385
XD ← RET-DESC
JALR
1
2
3
4
5
6
HANDLER: x86 TO RISC
EPC<1:0> == 00:
322
320
LR ← [SP]
SP ← SP + 4
AP ← SP
SP ← SP - 8
SP ← SP & (-32)
XD ← 0
EPC<1:0> == 01:
EPC<1:0> == 1x:
340
HANDLER: RISC TO x86
XD CONTAINS RETURN-DESCRIPTOR:
INTERPRET XD:
- REFORMAT/REPOSTION RESULT
- LOAD FPSW
SP ← [SP]//POP RA & ARGS
XD CONTAINS CALL-DESCRIPTOR:

FIG. 3M

09/434,374

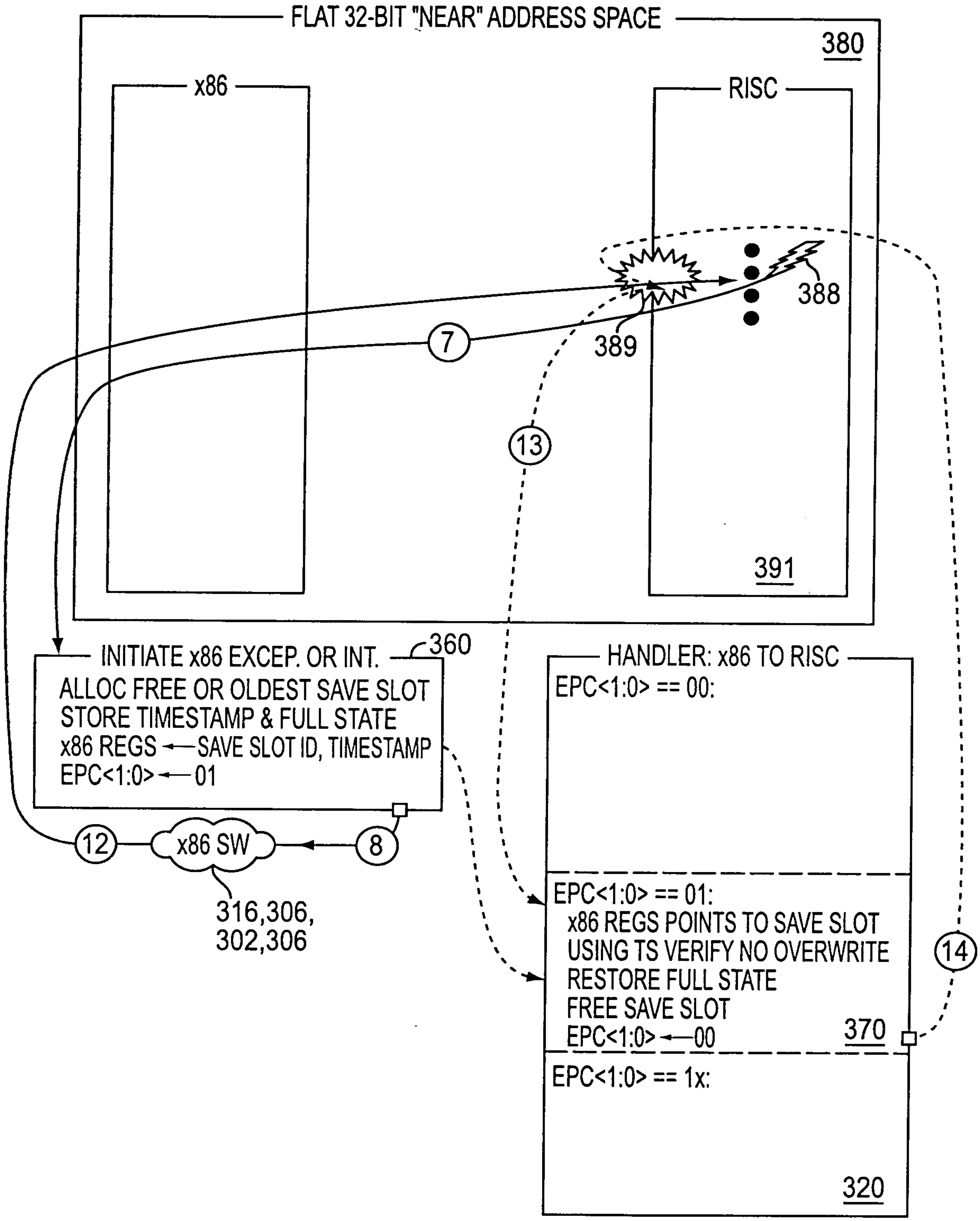


FIG. 3N

19/434,394

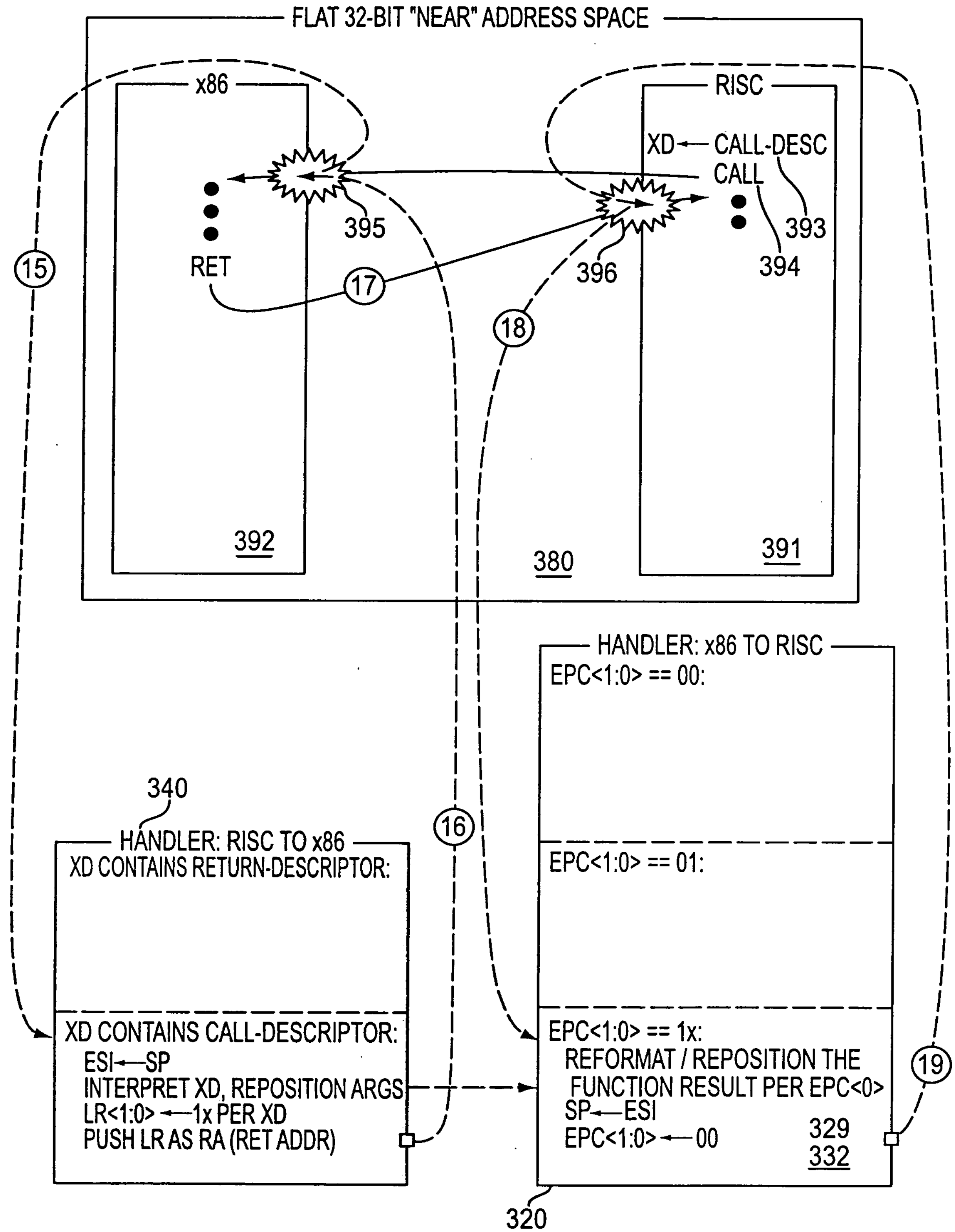
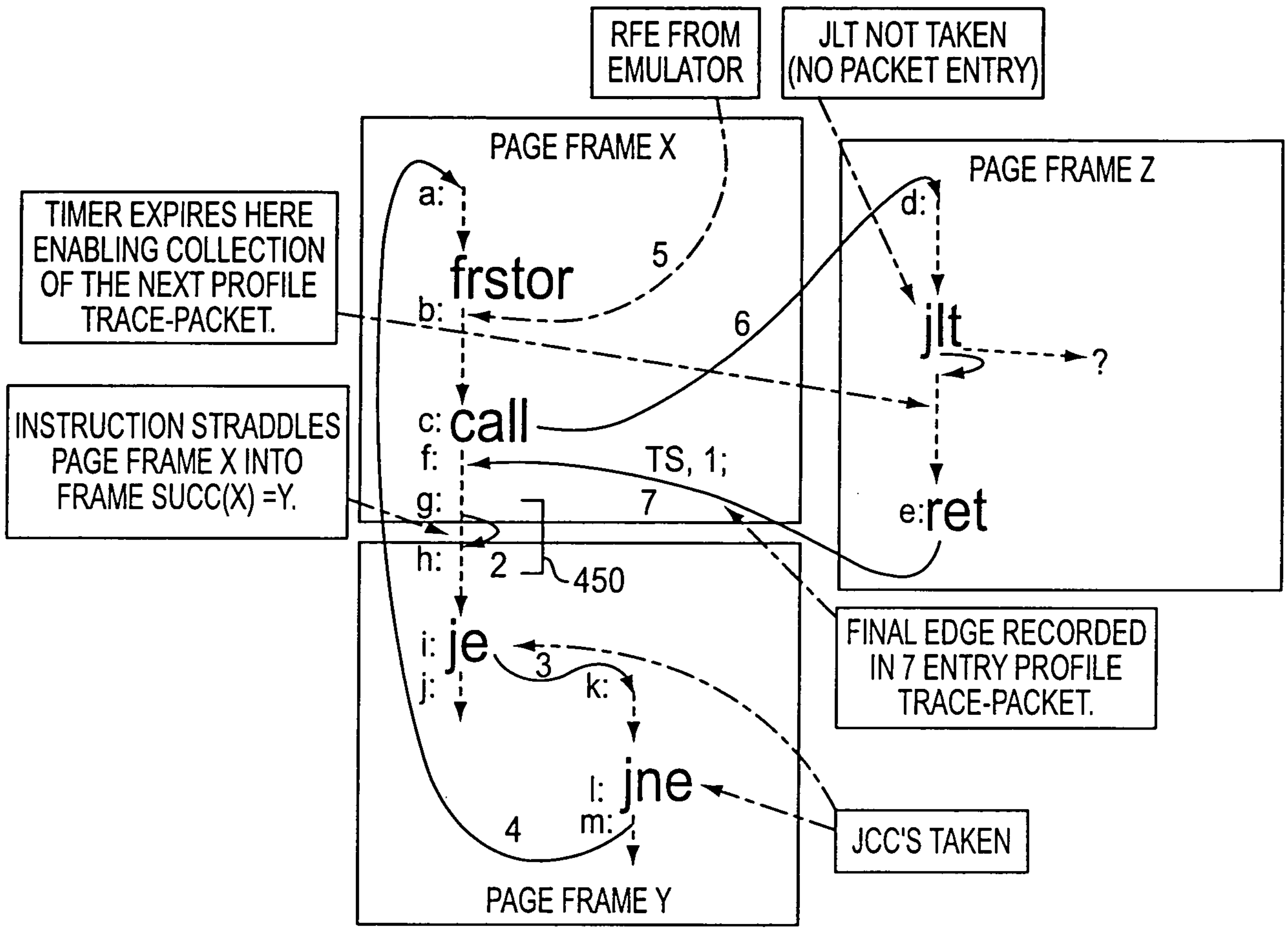


FIG. 3O

09/434,394



7 ENTRY TRACE PACKET (420)

ENTRY	EVENT CODE	DONE ADDR	NEXT ADDR	
	64 BIT TIME STAMP			
1	RET	x86 CONTEXT	*phys X:f*	430
2	NEW PAGE	*phys Y:g*	*phys Y:h*	440, 454
3	JCC FORWARD	*phys Y:i*	*phys Y:k*	440
4	JNZ BACKWARD	*phys Y:l*	*phys X:a*	440
5	SEQ; ENV CHANGE	x86 CONTEXT	*phys X:b*	430
6	IP-REL NEAR CALL	*phys X:c*	*phys Z:d*	440
7	NEAR RET	*phys Z:e*	*phys X:f*	440

FIG. 4A

09/434,394

SOURCE	CODE 402	EVENT	REUSE EVENT CODE 414	PROFILEABLE EVENT 416	INITIATE PACKET 418	PROBEABLE EVENT 610	PROBE EVENT BIT- ITLB PROBE ATTRIBUTE OR EMULATOR PROBE 612
410 RFE (CONTEXT_AT_POINT ENTRY); 412	0.0000	DEFAULT (x86 TRANSPARENT) EVENT, REUSE ALL CONVERTER VALUES	YES		NO		REUSE EVENT CODE
	0.0001	SIMPLE x86 INSTRUCTION COMPLETION (REUSE EVENT CODE)	YES		NO		REUSE EVENT CODE
	0.0010	PROBE EXCEPTION FAILED	YES		NO		REUSE EVENT CODE
	0.0011	PROBE EXCEPTION FAILED, RELOAD PROBE TIMER	YES		NO		REUSE EVENT CODE
	0.0100	FLUSH EVENT	NO	NO	NO	NO	-
	0.0101	SEQUENTIAL; EXECUTION ENVIRONMENT CHANGED - FORCE EVENT	NO	YES	NO	NO	-
	0.0110	FAR RET	NO	YES	YES	NO	-
	0.0111	IRET	NO	YES	NO	NO	-
	0.1000	FAR CALL	NO	YES	YES	YES	FAR CALL
	0.1001	FAR JMP	NO	YES	YES	NO	-
	0.1010	SPECIAL; EMULATOR EXECUTION, SUPPLY EXTRA INSTRUCTION DATA[a]	NO	YES	NO	NO	-
	0.1011	ABORT PROFILE COLLECTION	NO	NO	NO	NO	-
	0.1100	x86 SYNCHRONOUS/ ASYNCHRONOUS INTERRUPT W/PROBE (GRP 0)	NO	YES	YES	YES	EMULATOR PROBE
	0.1101	x86 SYNCHRONOUS/ASYNCHRONOUS INTERRUPT (GRP 0)	NO	YES	YES	NO	-
	0.1110	x86 SYNCHRONOUS/ASYNCHRONOUS INTERRUPT W/PROBE (GRP 1)	NO	YES	YES	YES	EMULATOR PROBE
	0.1111	x86 SYNCHRONOUS/ASYNCHRONOUS INTERRUPT (GRP 1)	NO	YES	YES	NO	-
404 CONVERTER (NEAR_EDGE ENTRY)	1.0000	IP-RELATIVE JNZ FORWARD (OPCODE: 75, OF 85)	NO	YES	YES	NO	-
	1.0001	IP-RELATIVE JNZ BACKWARD (OPCODE: 75, OF 85)	NO	YES	YES	YES	JNZ
	1.0010	IP-RELATIVE CONDITIONAL JUMP FORWARD - (JCC, JCXZ, LOOP)	NO	YES	YES	NO	-
	1.0011	IP-RELATIVE CONDITIONAL JUMP BACKWARD - (JCC, JCXZ, LOOP)	NO	YES	YES	YES	COND JUMP
	1.0100	IP-RELATIVE, NEAR JMP FORWARD (OPCODE: E9, EB)	NO	YES	YES	NO	-
	1.0101	IP-RELATIVE, NEAR JMP BACKWARD (OPCODE: E9, EB)	NO	YES	YES	YES	NEAR JUMP
	1.0110	RET/RET IMM16 (OPCODE C3, C2 /W)	NO	YES	YES	NO	-
	1.0111	IP-RELATIVE, NEAR CALL (OPCODE: E8)	NO	YES	YES	YES	NEAR CALL
	1.1000	REPE/REPNE CMPS/SCAS (OPCODE: A6, A7, AE, AF)	NO	YES	NO	NO	-
	1.1001	REP MOVS/STOS/LDOS (OPCODE: A4, A5, AA, AB, AC, AD)	NO	YES	NO	NO	-
	1.1010	INDIRECT NEAR JMP (OPCODE: FF /4)	NO	YES	YES	NO	-
	1.1011	INDIRECT NEAR CALL (OPCODE: FF /2)	NO	YES	YES	YES	NEAR CALL
	1.1100	LOAD FROM I/O MEMORY (TLB.ASI !=0) (NOT USED IN T1)	NO	YES	NO	NO	-
	1.1101	AVAILABLE FOR EXPANSION	NO	NO	NO	NO	-
	1.1110	DEFAULT CONVERTER EVENT; SEQUENTIAL 406	NO	NO	NO	NO	-
	1.1111	NEW PAGE (INSTRUCTION ENDS ON LAST BYTE OF A PAGE FRAME OR STRADDLES ACROSS A PAGE FRAME BOUNDARY) 408	NO	YES	NO	NO	-

FIG. 4B

09/434,394

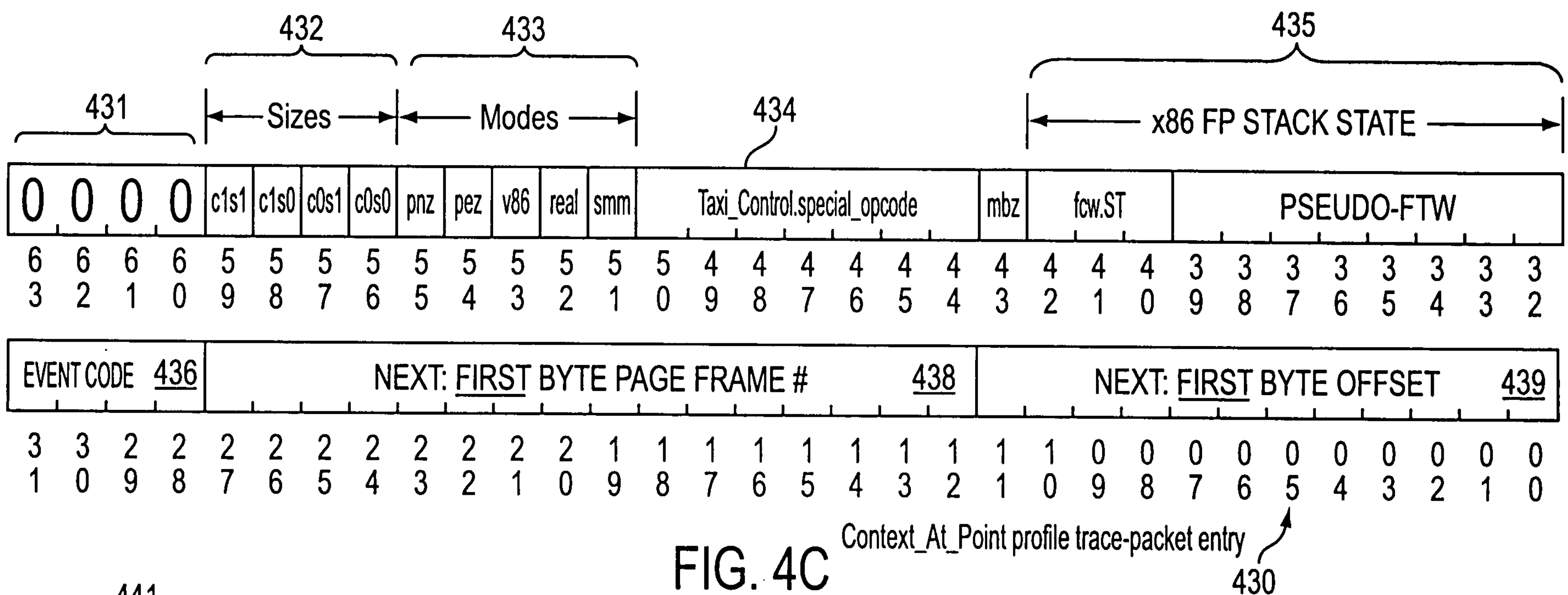


FIG. 4C

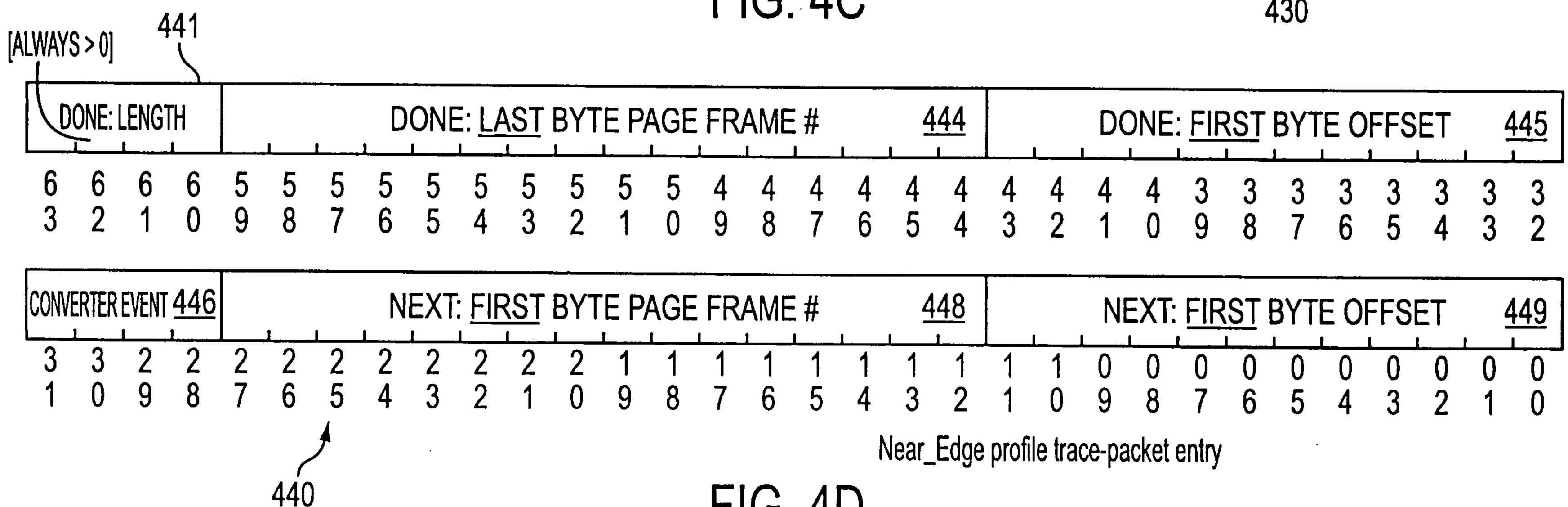


FIG. 4D

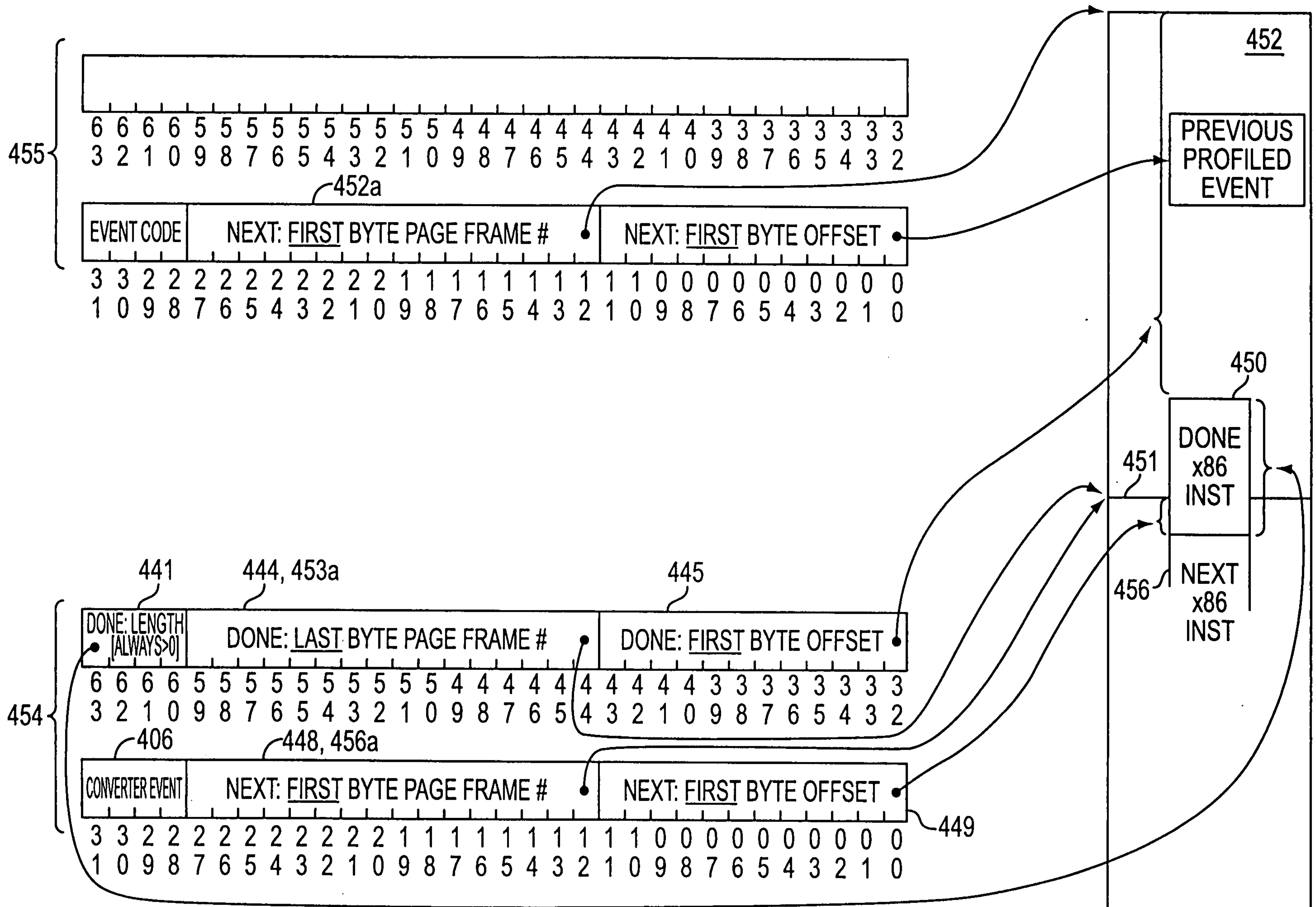


FIG. 4E

09/XX3XX,394

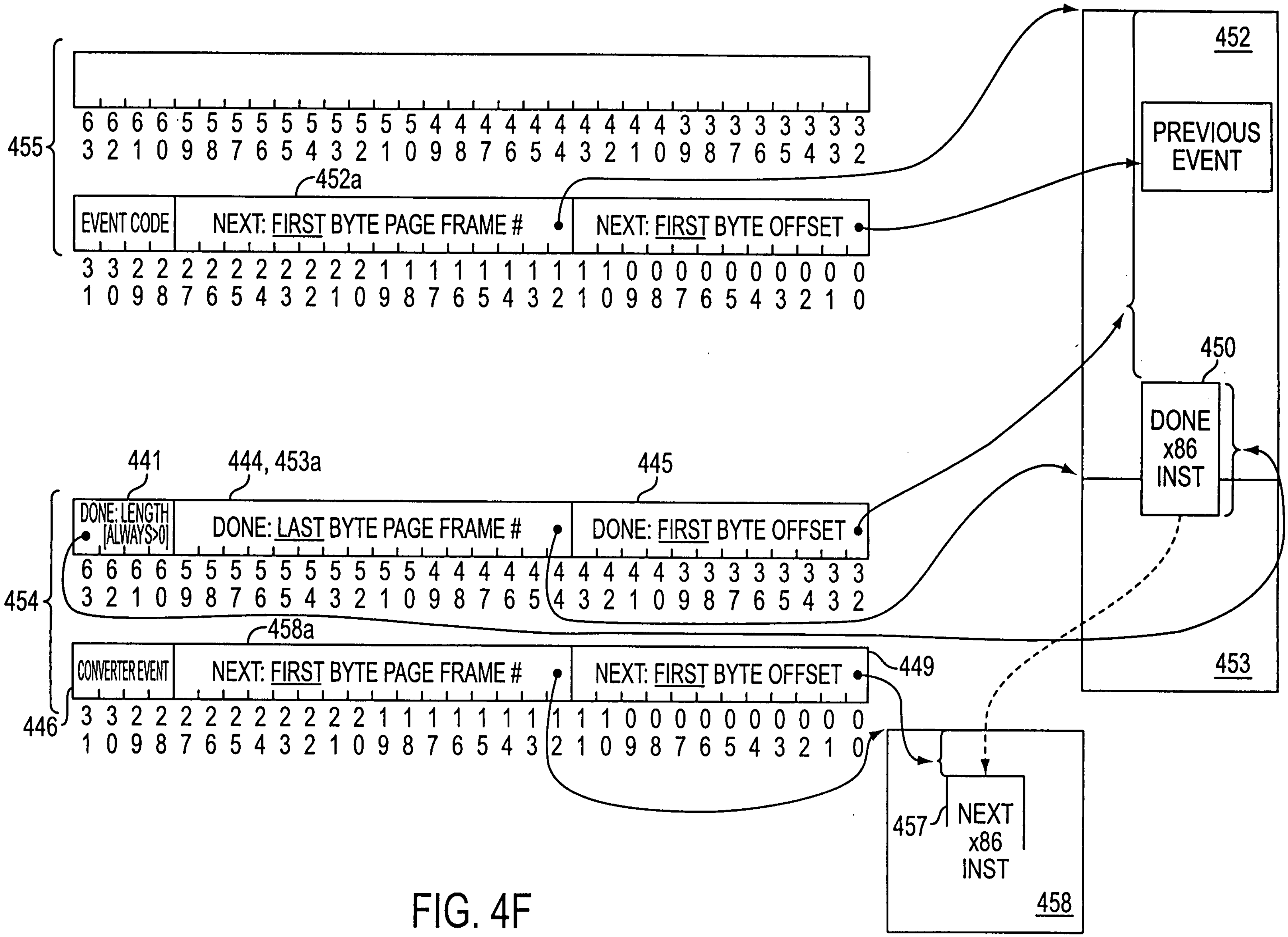


FIG. 4F

09/434,374

09/434,394

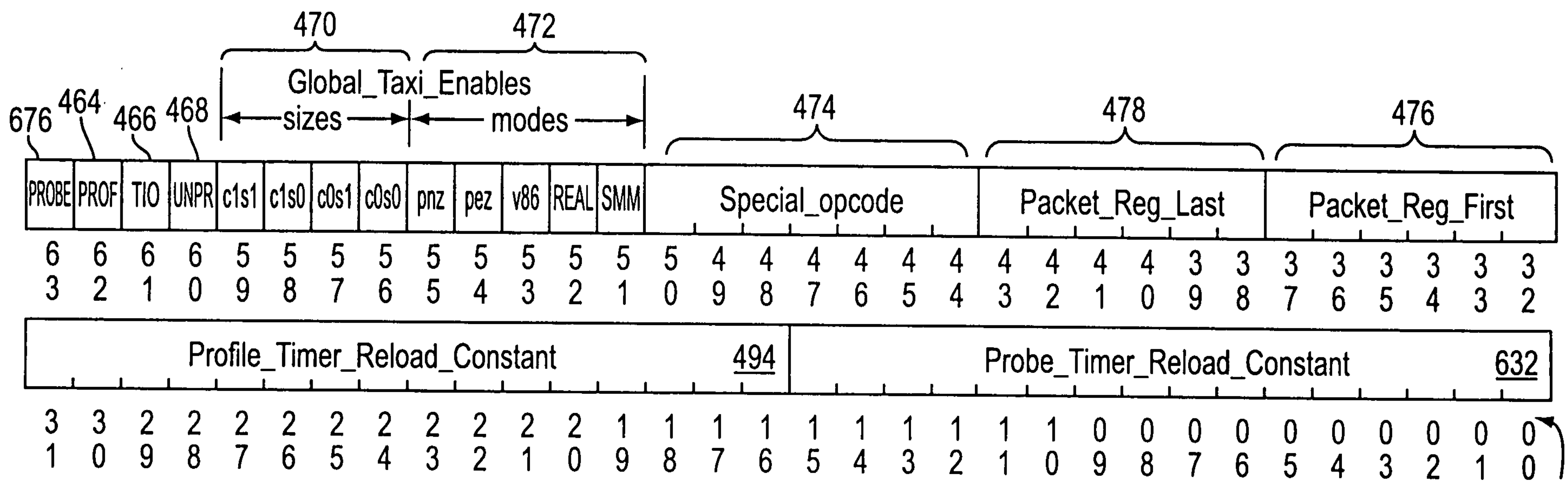


FIG. 4G

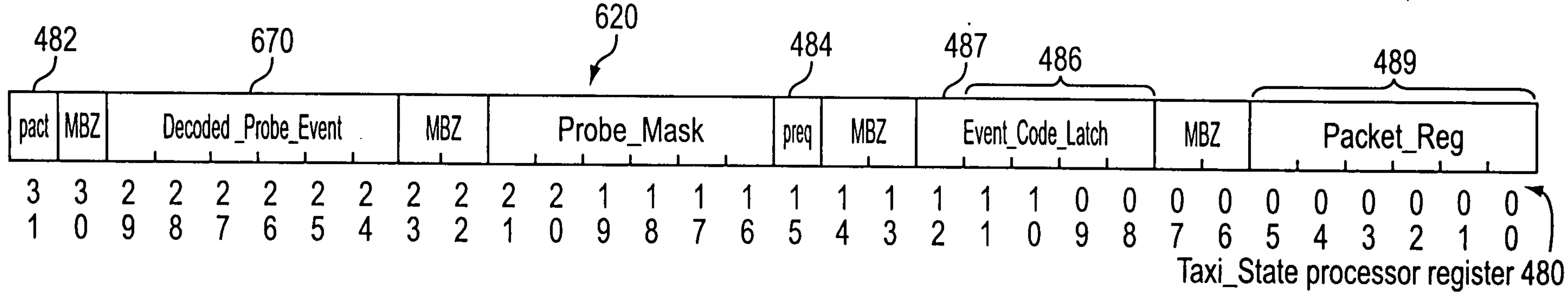


FIG. 4H

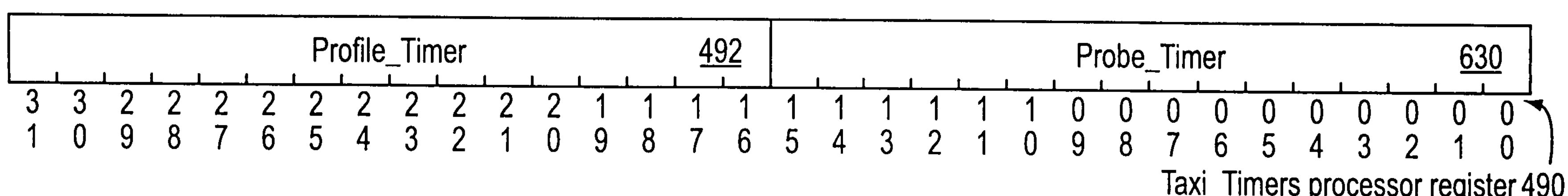


FIG. 4I

EVENTS
pe_{init} = "initiate packet" profile event 418
$pe_{\overline{init}}$ = non-"initiate packet" profile event
pe = any profile event 416
te = timer expiry
ap = abort packet

STATE VARIABLES
PR = Profile_Request flag 488
PA = Profile_Active flag 482

RULES
te EVENT:
PR ← 1

PR & $\overline{PA}$ & pe_{init}:
PR ← 0
PA ← 1
Init Packet_Reg
Save Timestamp
Log Event (CAP)
Full Packet? / Packet_Reg++

PA & pe:
Log Event (CAP or NE)
Full Packet? / Packet_Reg++

FULL PACKET:
PA ← 0
profile exception

ap EVENT:
PA ← 0

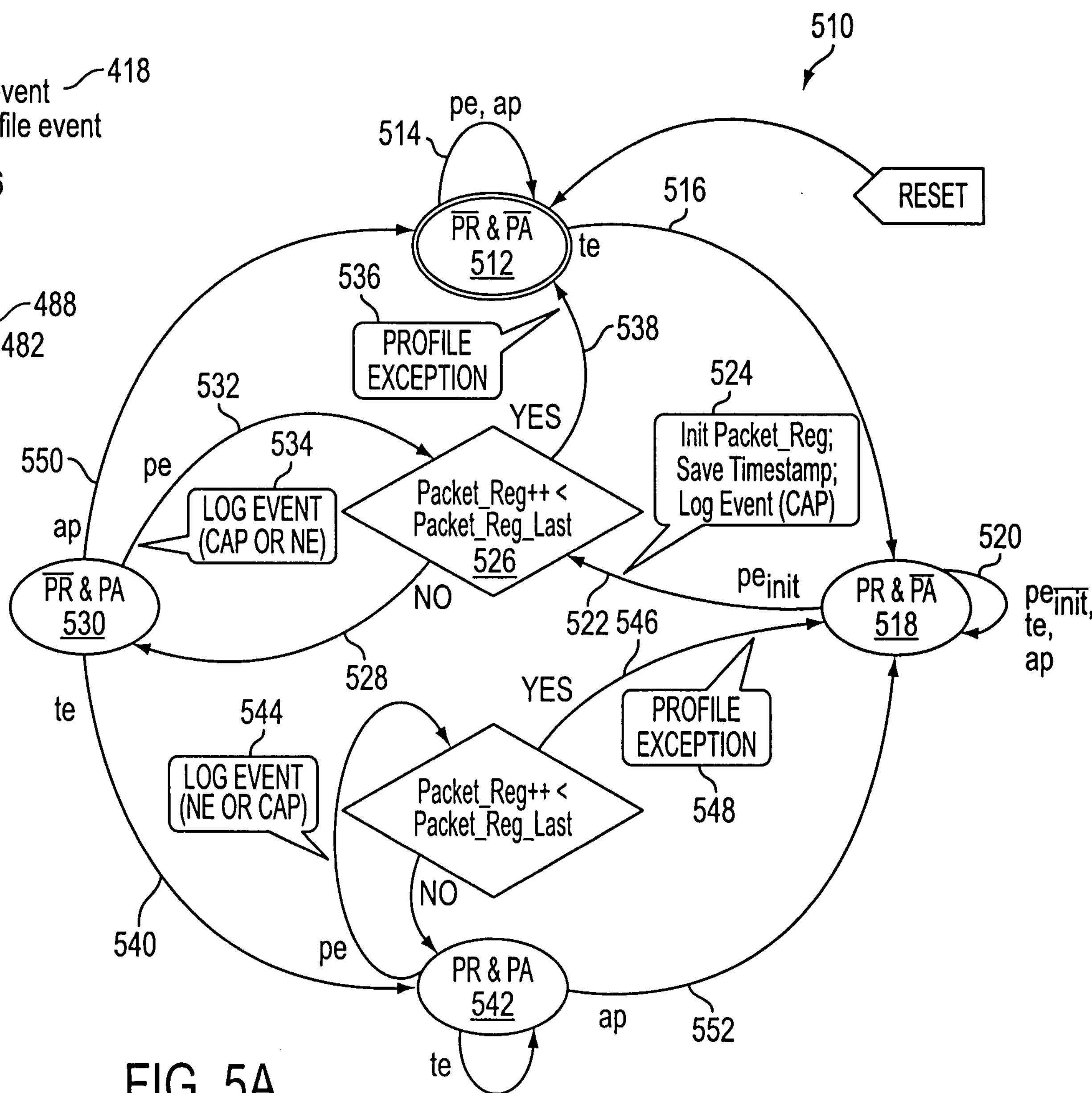


FIG. 5A

09/634,394

09/434,394

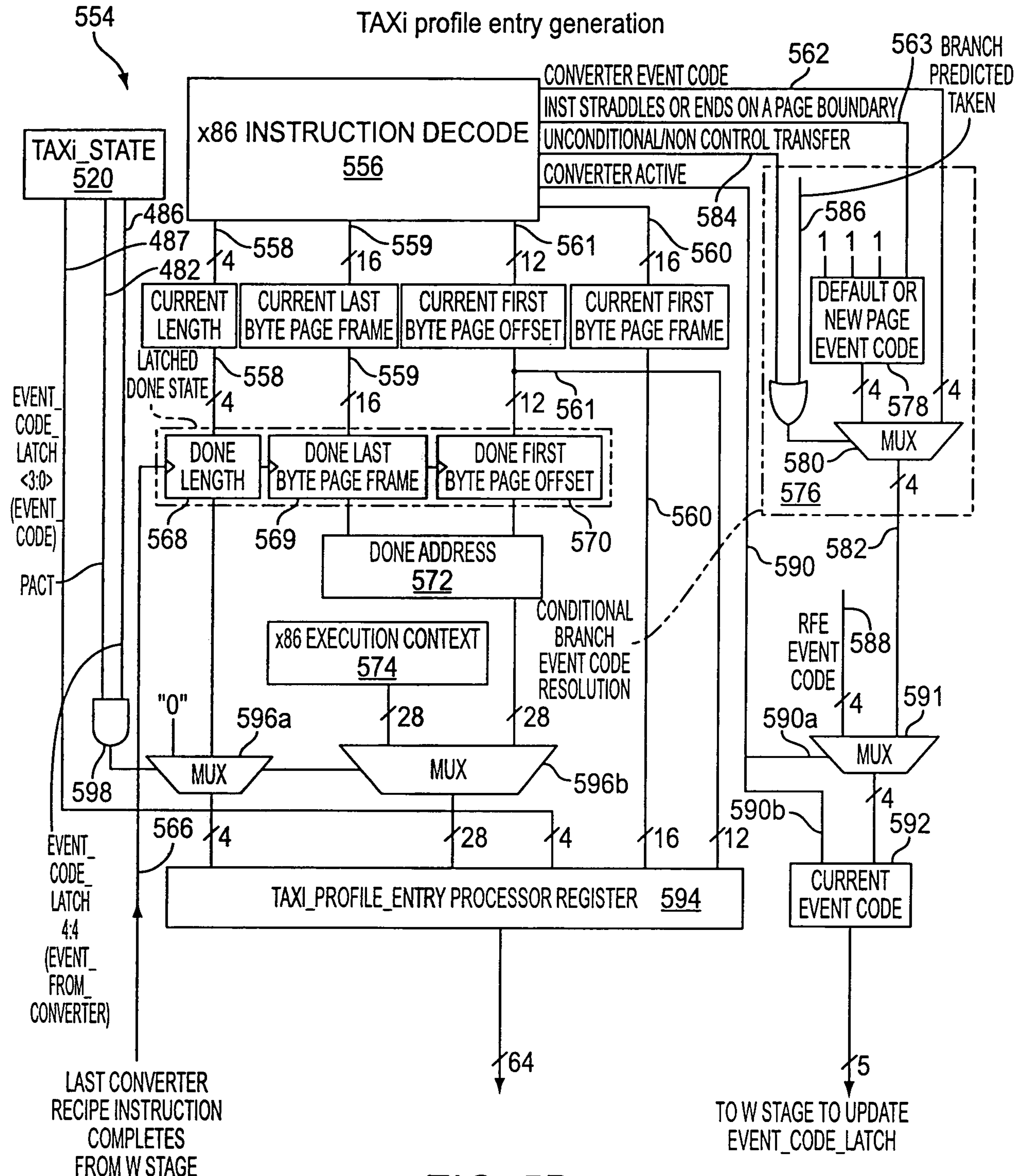


FIG. 5B

09/4 54, 574

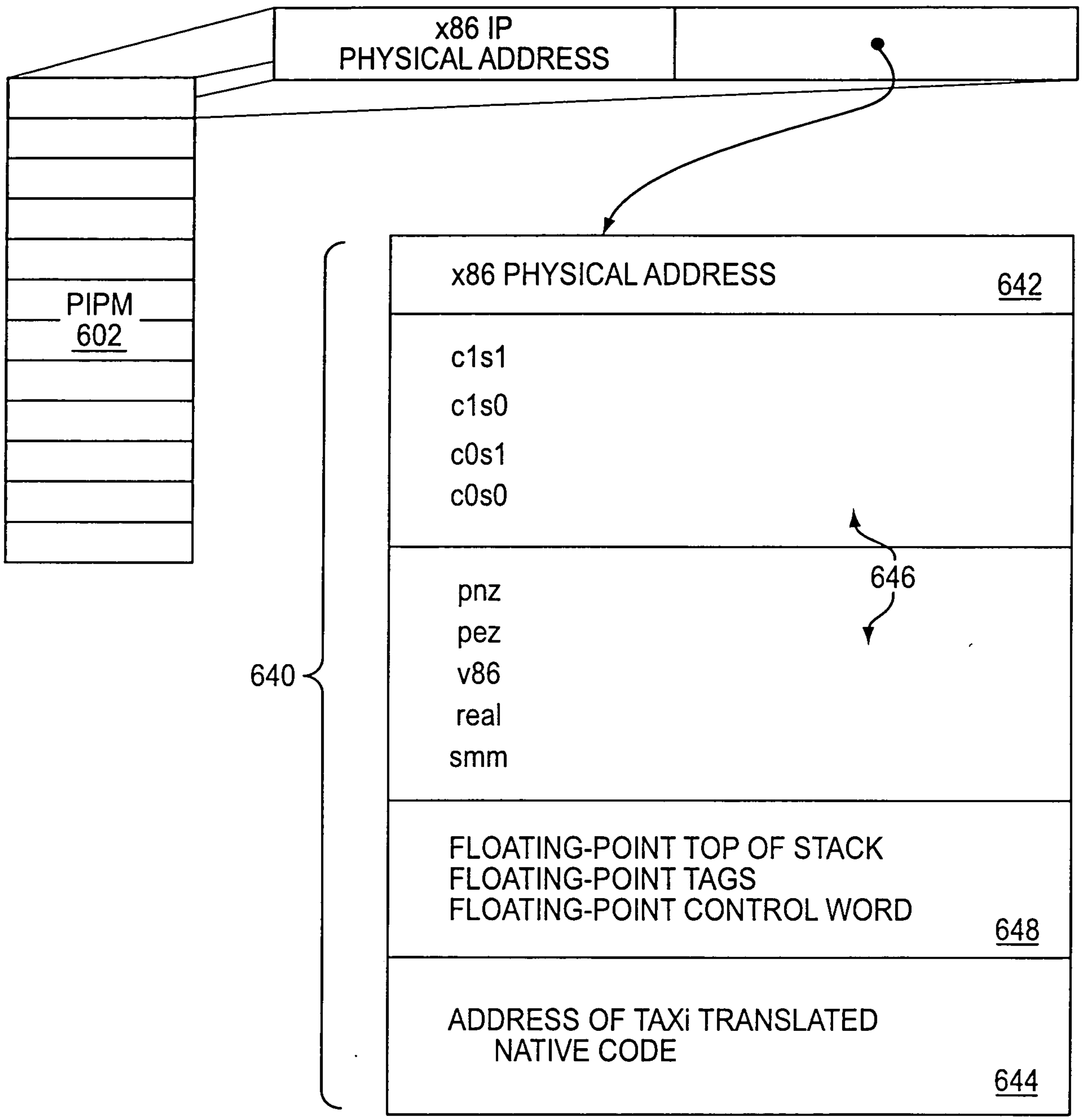


FIG. 6A

09/434,394

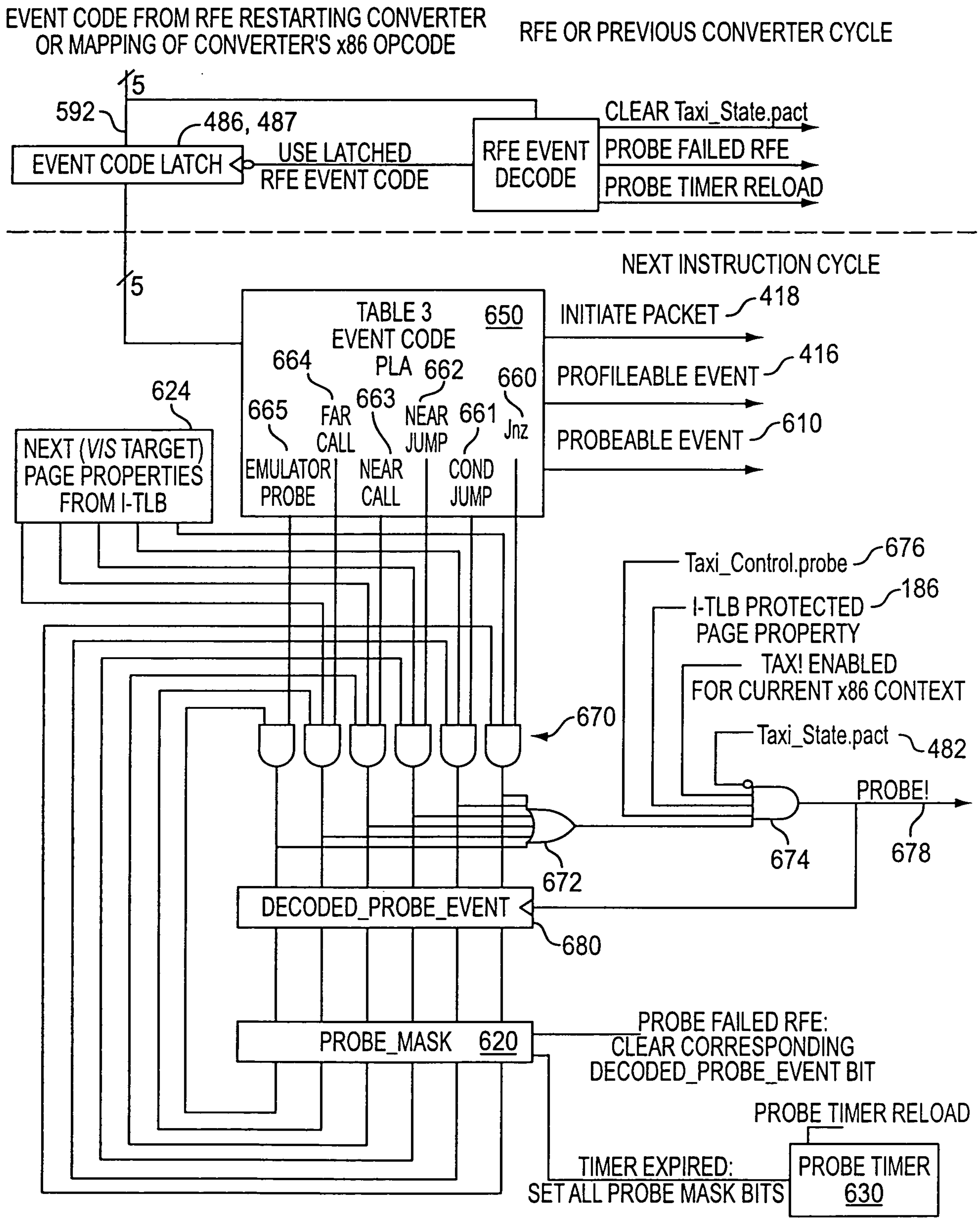


FIG. 6B

05/431,394

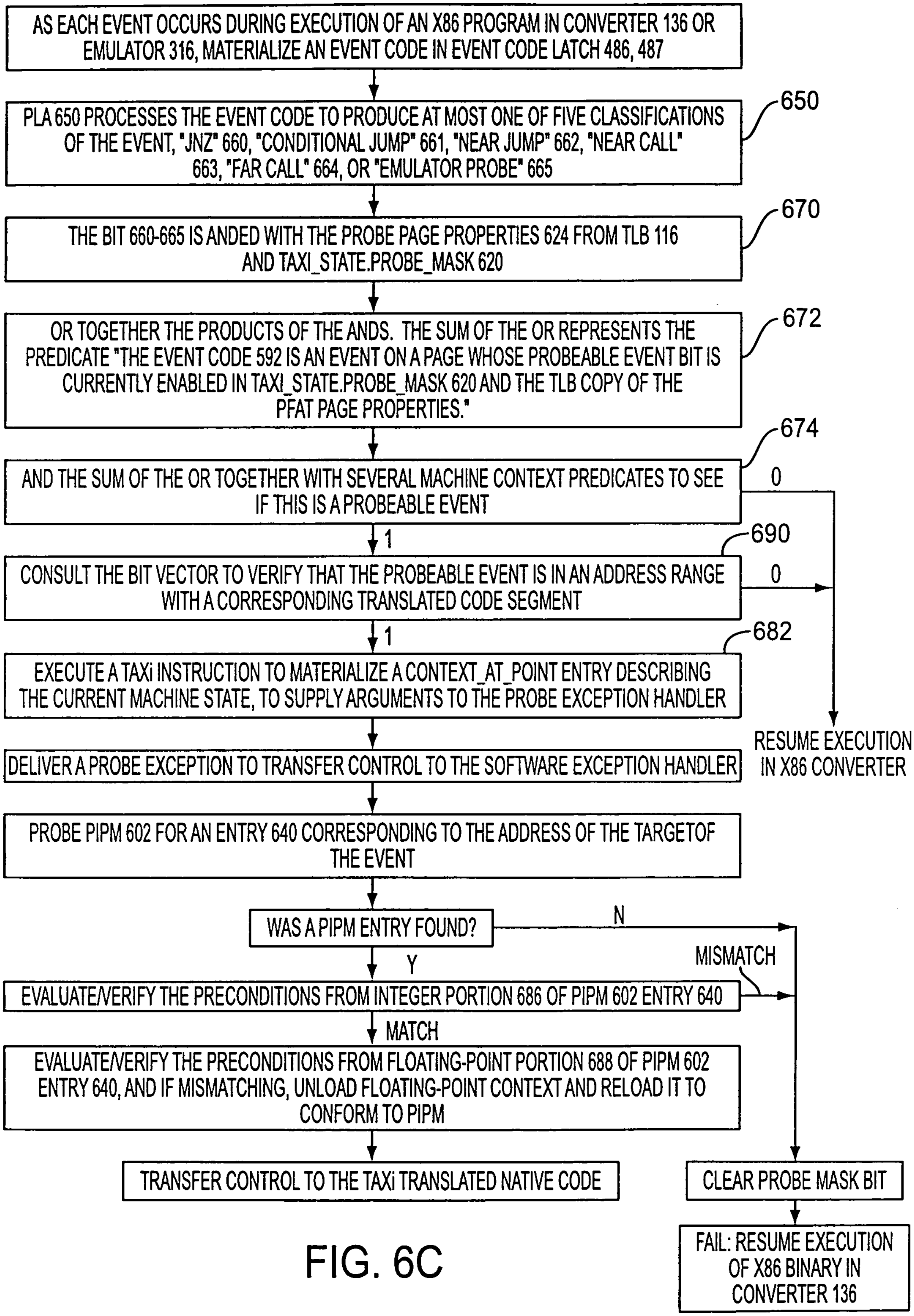


FIG. 6C

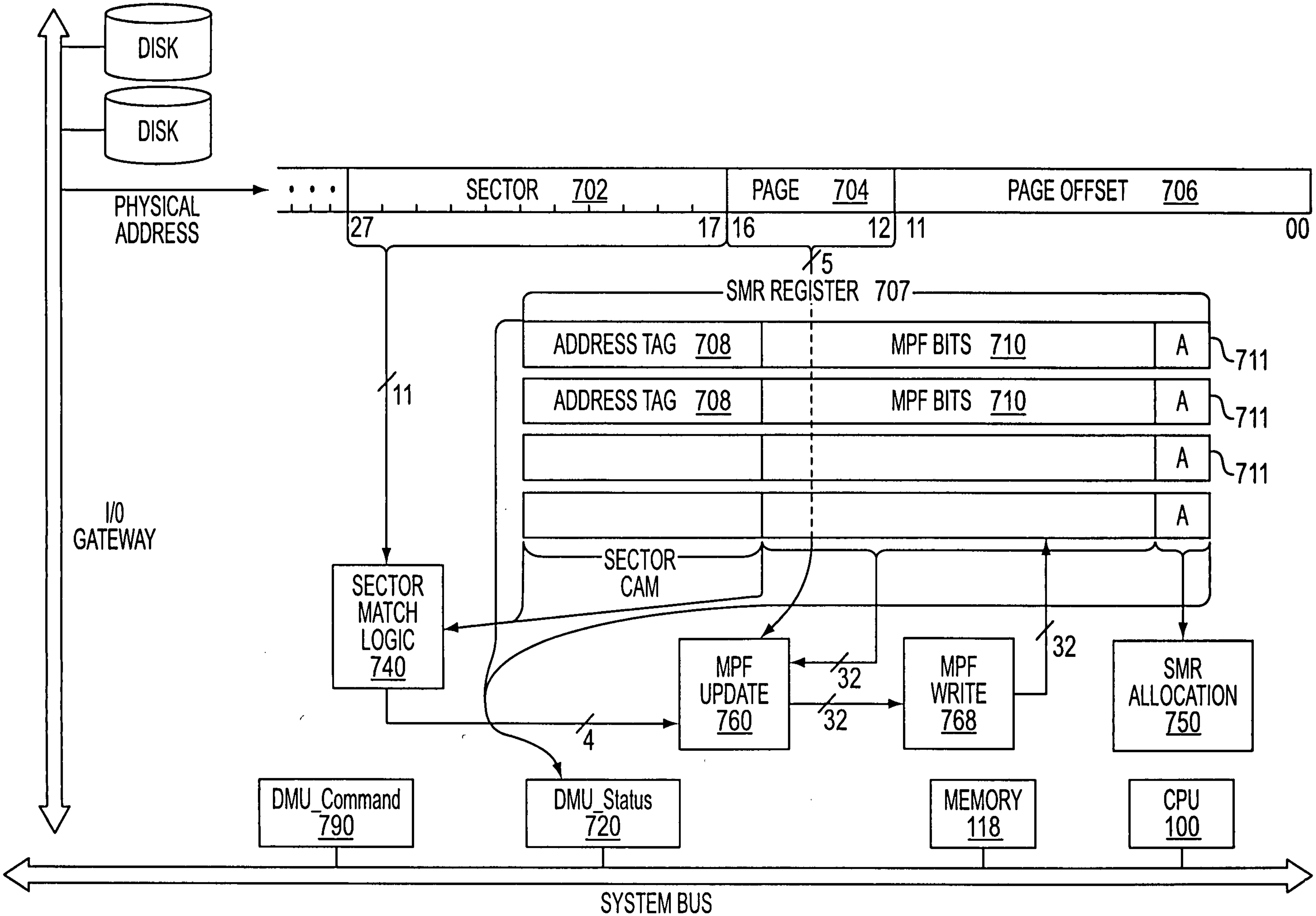


FIG. 7A

09/434,374

09/434,394

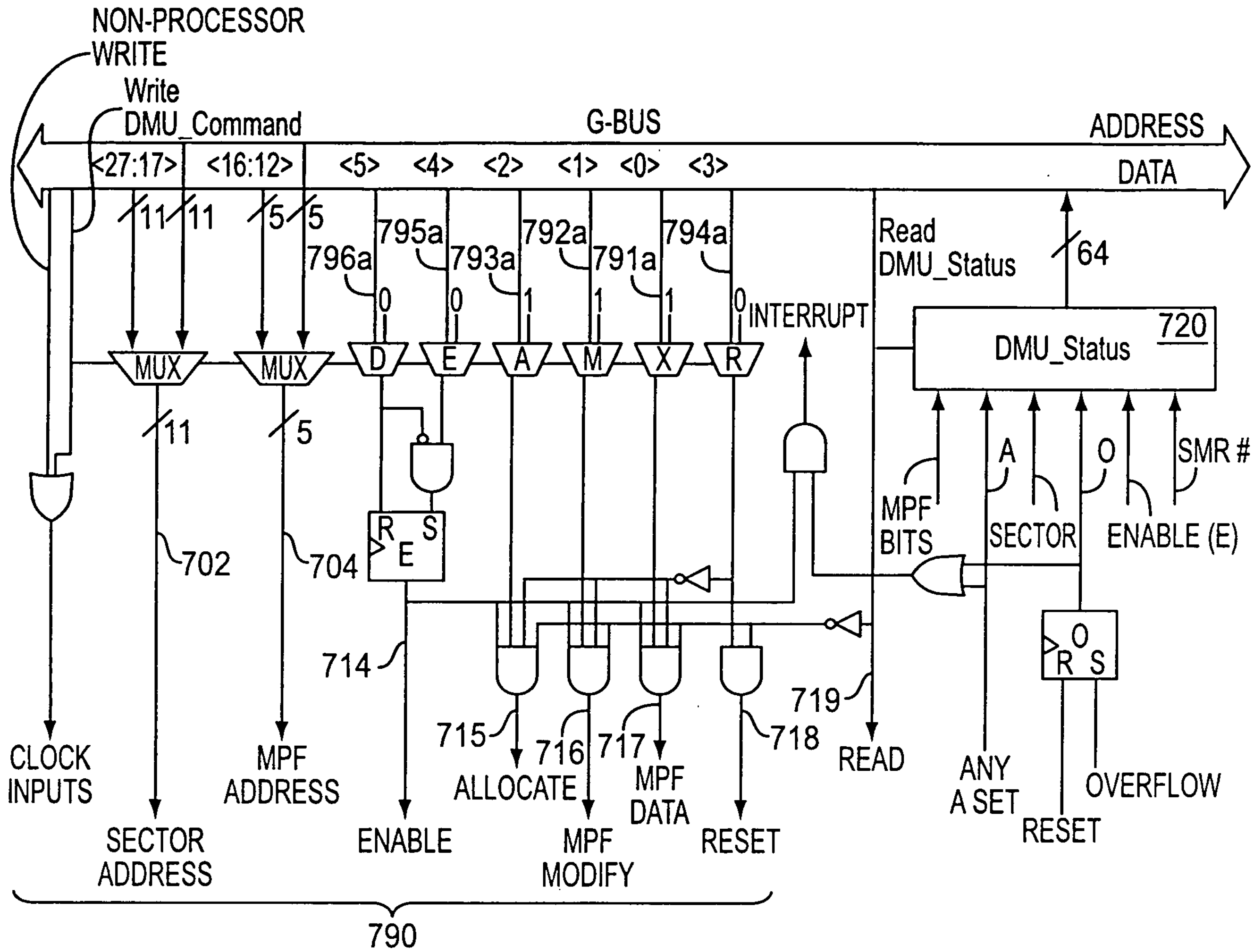


FIG. 7B

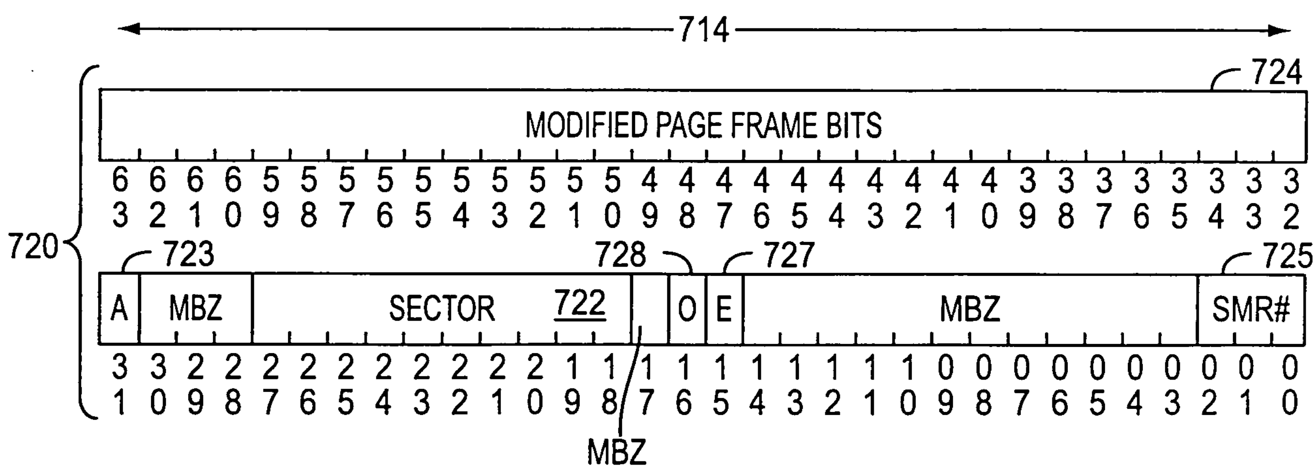


FIG. 7C

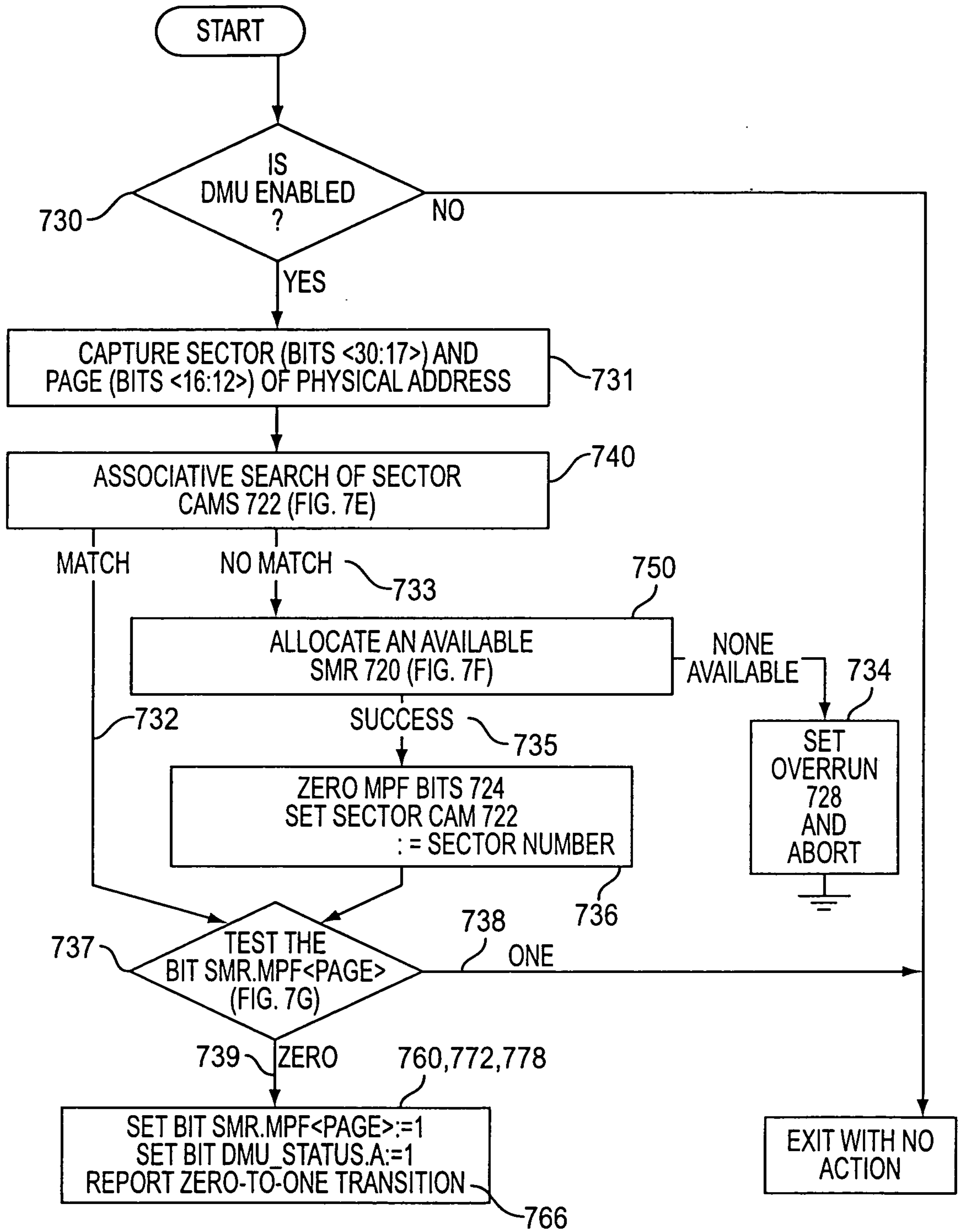
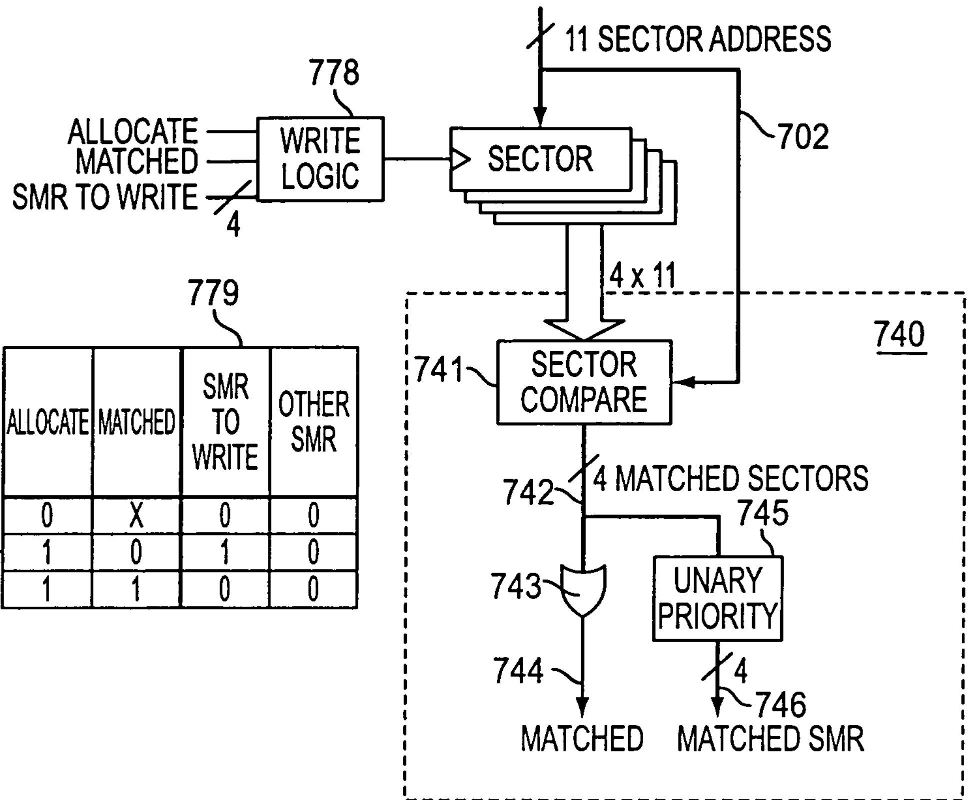

START
IS
DMU ENABLED
?
730
NO
YES
CAPTURE SECTOR (BITS <30:17>) AND
PAGE (BITS <16:12>) OF PHYSICAL ADDRESS
731
ASSOCIATIVE SEARCH OF SECTOR
CAMS 722 (FIG. 7E)
740
MATCH
NO MATCH
733
750
ALLOCATE AN AVAILABLE
SMR 720 (FIG. 7F)
NONE
AVAILABLE
734
732
SUCCESS
735
SET
OVERRUN
728
AND
ABORT
ZERO MPF BITS 724
SET SECTOR CAM 722
: = SECTOR NUMBER
736
738
737
TEST THE
BIT SMR.MPF<PAGE>
(FIG. 7G)
ONE
739
ZERO
760,772,778
SET BIT SMR.MPF<PAGE>:=1
SET BIT DMU_STATUS.A:=1
REPORT ZERO-TO-ONE TRANSITION
766
EXIT WITH NO
ACTION

FIG. 7D

09/434,344



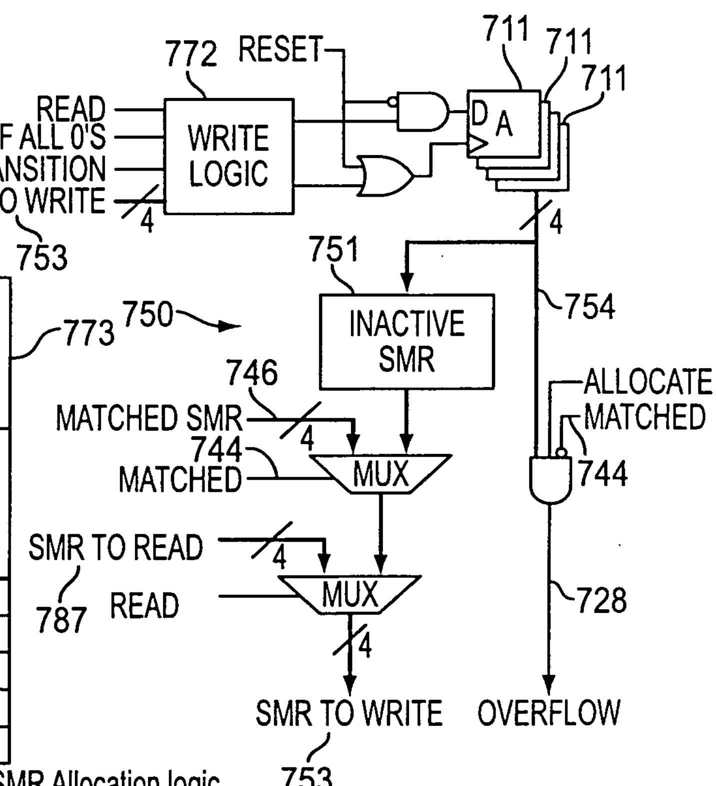
779

736

ALLOCATE	MATCHED	SMR TO WRITE	OTHER SMR
0	X	0	0
1	0	1	0
1	1	0	0

Sector match logic

FIG. 7E



773

719 READ	767 MPF ALL 0'S	0 TO 1 TRANSITION 766	SMR TO WRITE: WRITE 774	SMR TO WRITE: DATA 775	OTHER SMR: WRITE 776	OTHER SMR: DATA 777
0	0	0	0	X	0	X
0	0	1	1	1	0	X
0	1	0	1	0	0	X
0	1	1	-	-	-	-
1	X	X	1	0	0	X

SMR Allocation logic

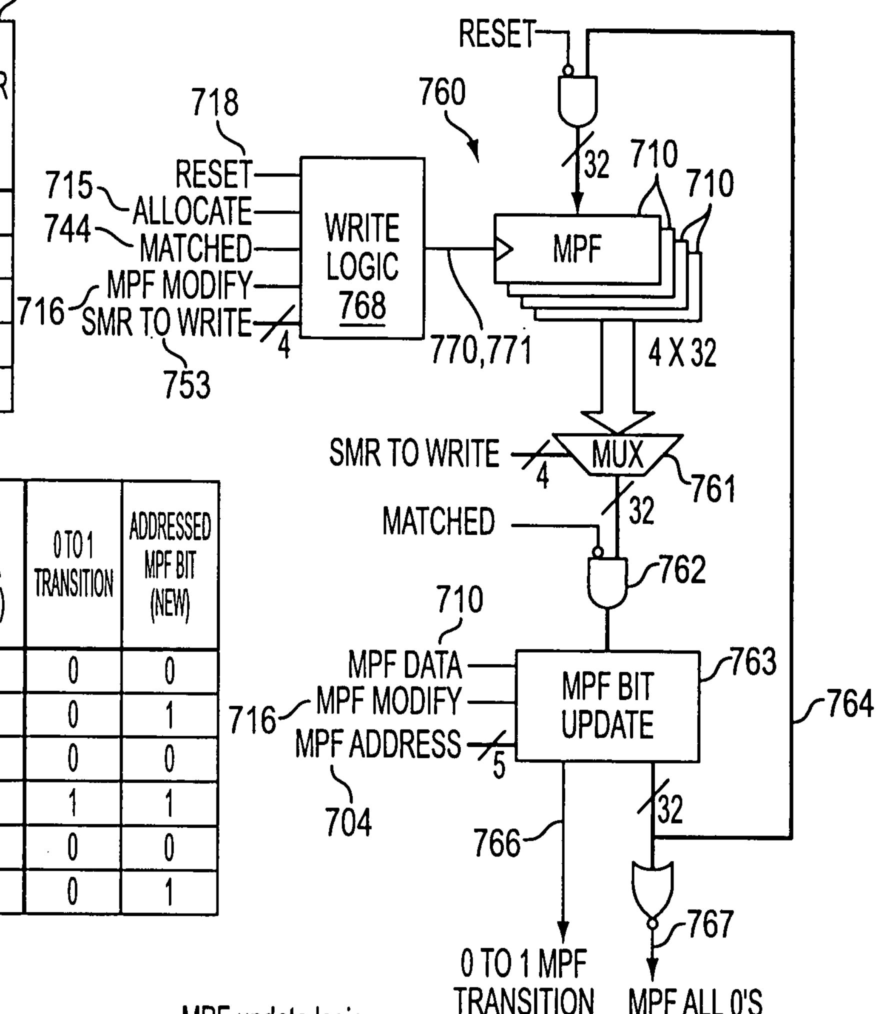
FIG. 7F

718	715	744	716	770	771
RESET	ALLOCATE	MATCHED	MPF MODIFY	SMR TO WRITE	OTHER SMR
0	X	X	0	0	0
0	0	0	1	0	0
0	X	1	1	1	0
0	1	X	1	1	0
1	X	X	X	1	1

736

765

MPF MODIFY	ADDRESSED MPF BIT (OLD)	MPF DATA (NEW)	0 TO 1 TRANSITION	ADDRESSED MPF BIT (NEW)
0	0	X	0	0
0	1	X	0	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	1

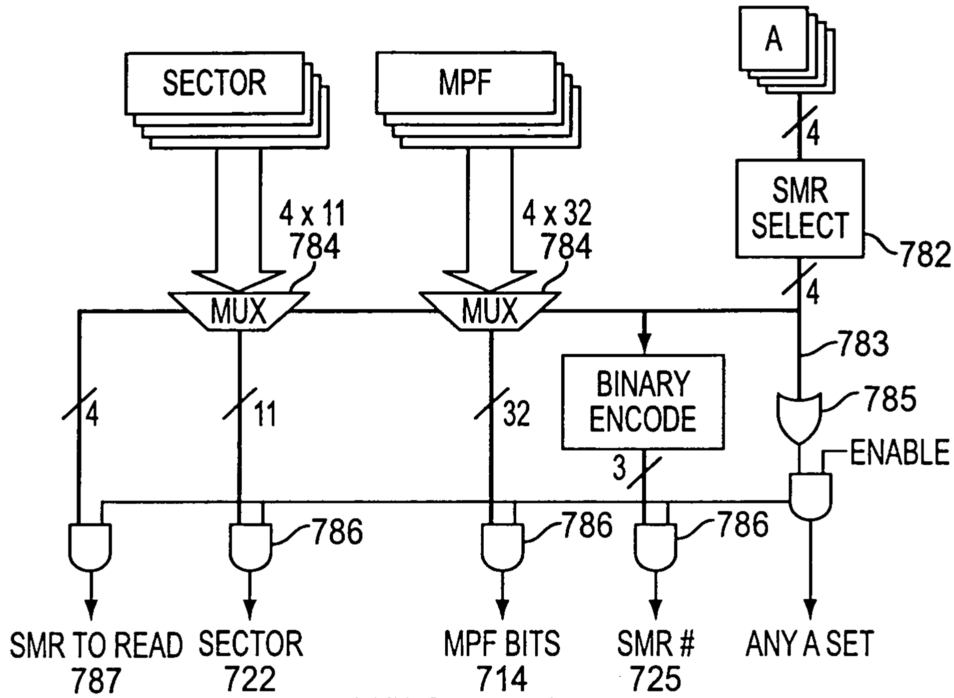


MPF update logic

FIG. 7G

0 9/434,394

09/434,394



DMU_Status read

FIG. 7H

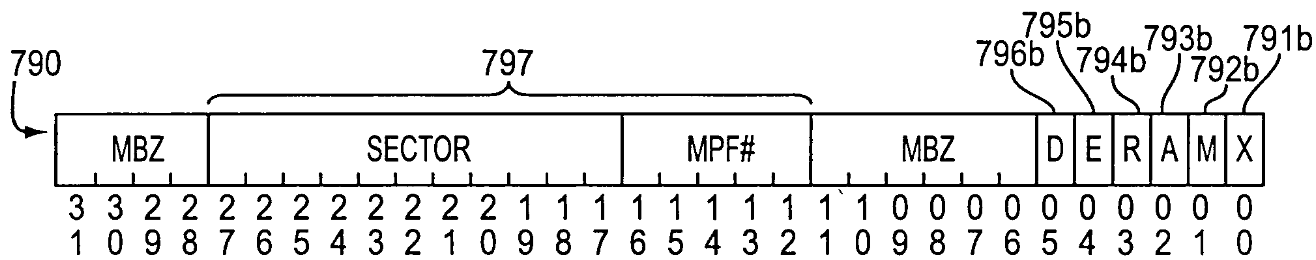


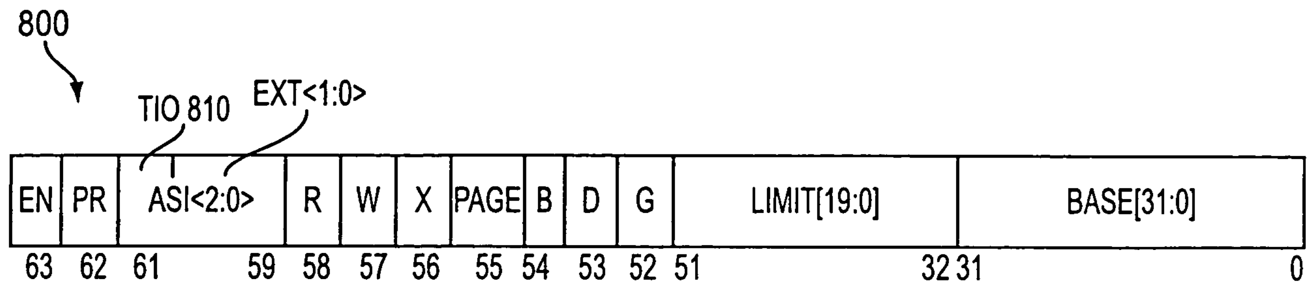
FIG. 7I

COMMAND BIT	BIT POSITION	MEANING
D	5	DISABLE MONITORING OF DMA WRITES BY CLEARING THE DMU ENABLE FLAG
E	4	ENABLE MONITORING OF DMA WRITES BY SETTING THE DMU ENABLE FLAG
R	3	RESET ALL SMRS: CLEAR ALL A AND MPF BITS AND CLEAR THE DMU OVERRUN FLAG
A	2	ALLOCATE AN INACTIVE SMR ON A FAILED SEARCH
M	1	ALLOW MPF MODIFICATIONS
X	0	NEW MPF BIT VALUE TO RECORD ON SUCCESSFUL SEARCH (OR ALLOCATION)

M	X	ACTION
0	-	INHIBIT MODIFICATION OF THE MPF BIT
1	0	CLEAR THE CORRESPONDING MPF BIT
1	1	SET THE CORRESPONDING MPF BIT

FIG. 7J

09/334,394



SIZE	BIT(S)	NAME	FUNCTION
1	63	SEG.EN	ENABLES SEGMENT LIMIT/PROTECTION CHECKING
1	62	SEG.PR	CHOOSES WHICH PROTECTION BITS TO USE FOR PAGE TABLE PROTECTION - (0 MEANS PSW.UK OR 1 MEANS MISC.UK)
3	61:59	SEG.AS	ADDRESS SPACE (ONLY USED WHEN SEG.PAGE IS 0)
		SEG.TIO, SEG.EXT	ADDRESS SPACE EXTENSION (ONLY USED WHEN SEG.PAGE IS 1)
3	58:56	SEG.RWX	READ/WRITE/EXECUTE '1' MEANS ENABLED - ALL 000 MEANS IT'S AN INVALID SEGMENT
1	55	SEG.PAGE	ENABLES THE PAGING SYSTEM -- (TRANSLATION AND CHECKING)
1	54	SEG.B	SEGMENT SIZE (1 MEANS 32-BIT, 0 MEANS 16-BIT)
1	53	SEG.D	SEGMENT DIRECTION (0 MEANS EXPAND UP)
1	52	SEG.G	SIZE OF LIMIT (1 MEANS IT'S IN 4k PAGES)
20	51:32	SEG.LIMIT	SEGMENT LIMIT
32	31:0	SEG.BASE	SEGMENT BASE

FIG. 8A

09/434,394

AT CODE GENERATION TIME:

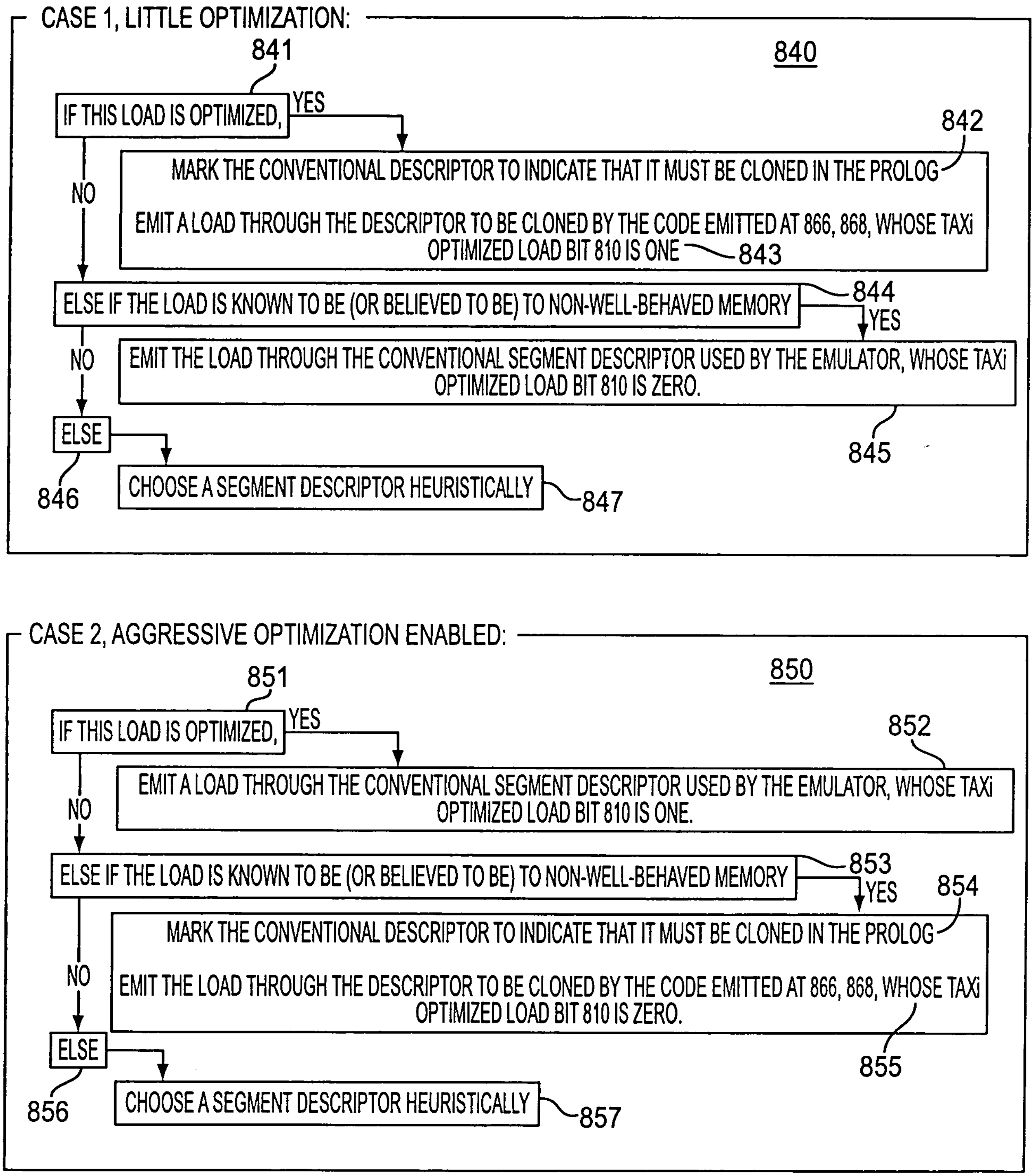


FIG. 8B

09/434,354

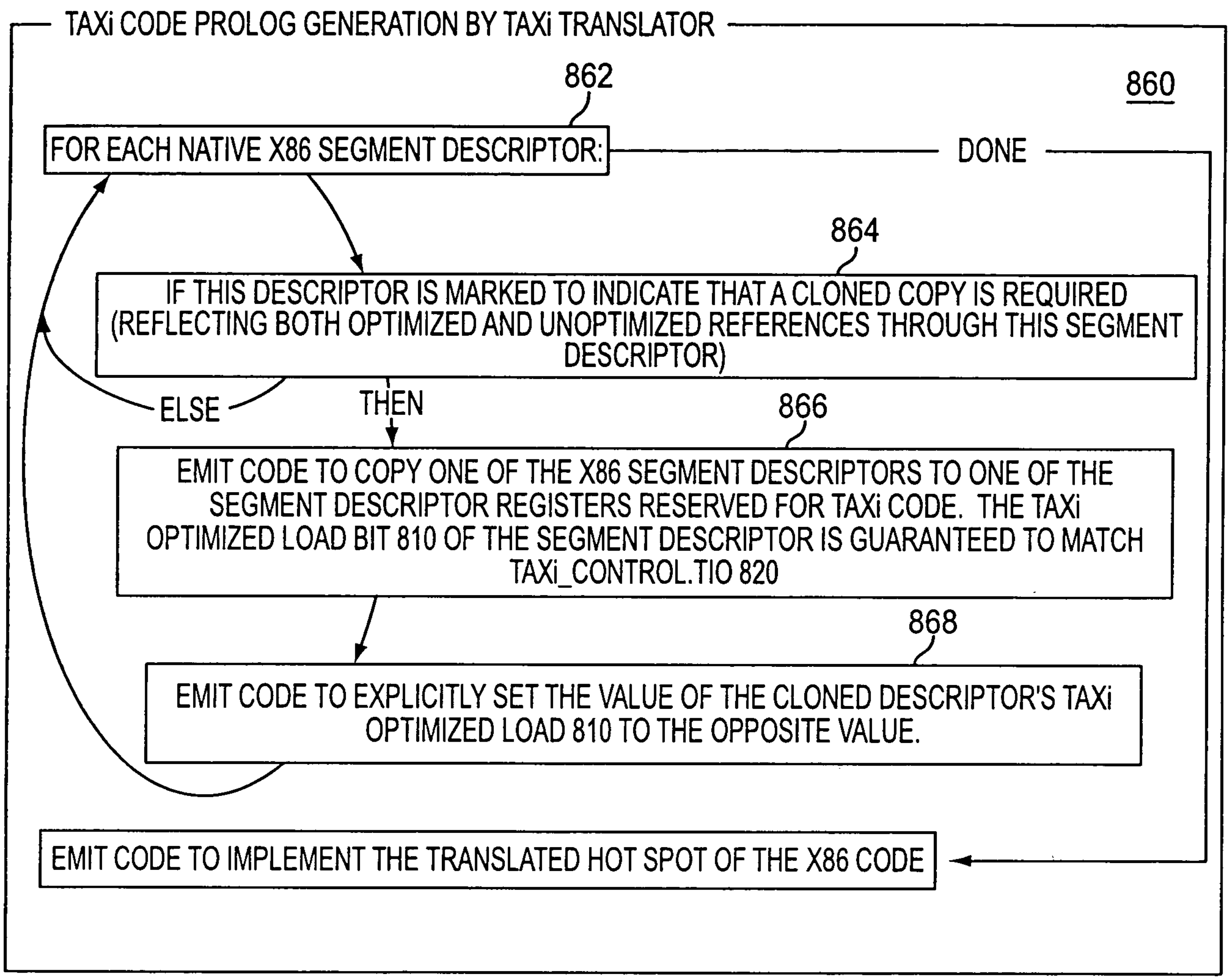


FIG. 8C